VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

WEBOVÝ PORTÁL PRO APLIKACI METODIK PRO ZVYŠOVÁNÍ SPOLEHLIVOSTI

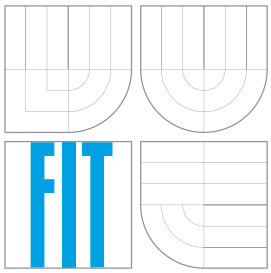
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR POUPĚ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

WEBOVÝ PORTÁL PRO APLIKACI METODIK PRO ZVYŠOVÁNÍ SPOLEHLIVOSTI

WEB PORTAL FOR FAULT TOLERANT METHODOLOGY APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR POUPĚ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ MIČULKA

BRNO 2012

Abstrakt

Diplomová práce se zabývá vývojem webového portálu pro aplikaci metodik pro zvýšení spolehlivosti. Uvádí do problematiky systémů odolných proti poruchám a analyzuje požadavky na systém, které mají uživatelé pracující v tomto oboru. Popisuje cyklus vývoje od analýzy a specifikace aplikace přes návrh systému až po část implementace a testování. Detailněji se zaměřuje na návrh portálu, který nabízí komplexní a univerzální řešení problému, které vede k výsledné realizaci tohoto portálu. Tato realizace je pak součástí práce.

Abstract

This master's thesis deals with the development of web portal for the application of fault–tolerant methodologies. It introduces the issue of fault–tolerant systems and analyze system requirements, that have users working in this field. It describes the development cycle from analysis and specification of application system design through to implementation and testing part. More thoroughly it is focusing above design portal that provides a comprehensive and versatile solution to the problem that leads to the final implementation of this portal. This realization is part of the thesis.

Klíčová slova

Webový portál, hlídací obvod, systém odolný proti poruchám, spolehlivost, PHP, Nette, jQuery, Graphviz, C/C++.

Keywords

Web portal, checker design, fault–tolerant system, dependibality, PHP, Nette, jQuery, Graphviz, C/C++.

Citace

Petr Poupě: Webový portál pro aplikaci metodik pro zvyšování spolehlivosti, diplomová práce, Brno, FIT VUT v Brně, 2012

Webový portál pro aplikaci metodik pro zvyšování spolehlivosti

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Lukáše Mičulky.

.....
Petr Poupě
21. května 2012

Poděkování

Mé poděkování patří Ing. Lukáši Mičulkovi za přínosné konzultace problémů a především mé rodině za podporu při studiu.

Obsah

1 Úvod **4**

2 Námět pro vznik portálu **6**
2.1 Současné možnosti vývoje 6
2.2 Možnosti změny 6

3 Architektury odolné proti poruchám **8**
3.1 Metody pro zajištění odolnosti architektury 8
3.1.1 TMR 9
3.1.2 Duplex 9
3.2 Hlídací obvody 10
3.3 Aplikace hlídacích obvodů 10
3.4 Dostupné nástroje pro tvorbu FT systému 11

4 Analýza a specifikace požadovaného systému **12**
4.1 Účel aplikace 12
4.1.1 Shromažďování spustitelných souborů 12
4.1.2 Přístup k programům 13
4.1.3 Zobrazení projektu 14
4.2 Možnosti zobrazování bloků 15
4.3 Bezpečnost systému 16
4.3.1 Spustitelné soubory na serveru 17
4.4 Uživatelské rozhraní 17
4.4.1 Obsluha a návštěvníci portálu 17

5 Návrh systému **19**
5.1 Diagram případů použití 19
5.2 Rozvržení tvorby bloku a projektu 21
5.2.1 Schéma tvorby bloku 21
5.2.2 Schéma tvorby projektu 24
5.3 Rozdělení aplikace 27
5.4 PHP aplikace 27
5.4.1 Uchovávání interních dat 27
5.4.2 Adresářová struktura pro podporu databáze 29
5.4.3 Tvorba grafu a vyhledávání 31
5.5 Spouštěcí programy v C++ 33
5.5.1 Runner 34
5.5.2 Deamon 35

6 Implementace **37**
6.1 C/C++ . . . 37
6.2 PHP . . . 37
6.2.1 Framework Nette . . . 38
6.3 HTML . . . 38
6.4 CSS . . . 38
6.5 MySQL . . . 39
6.6 JavaScript . . . 39

7 Testování **40**
7.1 Webová aplikace . . . 40
7.2 Spouštěč . . . 41
7.3 Konfigurace pro spuštění . . . 41

8 Závěr **42**
8.1 Dosažené výsledky . . . 42
8.2 Možnosti dalšího vývoje . . . 43

A Uživatelův průvodce portálem **47**

B Obsah CD **54**

Seznam obrázků

3.1 Schéma systému TMR. 9
3.2 Schéma techniky zdvojení se srovnávacím členem. 9
3.3 Duplex architektura s využitím hlídacích obvodů. 10
3.4 TMR architektura s využitím hlídacích obvodů. 10

4.1 Výskyt kružnice, vedoucí k uváznutí. 15
4.2 Korektní situace nevedoucí k uváznutí. 15
4.3 Ukázkový výstup programu `dot`. 16

5.1 Diagram případů užití. 20
5.2 Schéma postupu při tvorbě bloku. 21
5.3 Schéma postupu při tvorbě projektu. 24
5.4 Entity-relationship diagram. 28
5.5 Adresářová stuktura pro ukládání souborů. 30
5.6 Diagram aktivit pro program *Runner*. 34

A.1 Vzhled aplikace s přihlašovací obrazovkou. 47
A.2 Výpis vytvořených bloků. 48
A.3 Přidávání programů do bloku. 49
A.4 Editace parametrů v bloku. 50
A.5 Posun programů v bloku. 51
A.6 Výpis veřejných a vlastních projektů. 52
A.7 Vzhled aplikace s přihlašovací obrazovkou. 52
A.8 Schéma projektu. 53
A.9 Výstup po spuštění projektu. 53

Kapitola 1

Úvod

„Vše, co se porouchat může, to se porouchá!“

To tvrdí jeden z Murphyho zákonů a každý z nás má jistě dostatek důkazů o tom, že to platí pro každý obor lidské činnosti. Zůstává jen otázkou kdy.

V roce 1991 bylo v Perském zálivu zabito 28 lidí a 97 zraněno díky drobné chybě. Řídicí systém obraných raket Patriot pracoval bez přestávky více než 100 hodin, na což nebyl testován. Při zaokrouhlování hodin během ukládání do paměti došlo k poměrně malé odchylce oproti reálnému času. Díky tomu však radar hledal střelu při útoku téměř o 700 metrů dál, než skutečně byla, a systém neměl šanci uspět při obraně vojenské základny.

V dnešní době je náš civilizovaný život závislý na strojích všeho druhu. Ať už jsou to stroje mechanické, či elektrické, spoléháme na ně natolik, že v případě jejich poruchy dochází ke ztrátě financí, poškození životního prostředí nebo při nejhorším ke ztrátám na životech. Mělo by tedy být samozřejmostí, že spolehlivost dostatečně zajistíme, ale děje se tak i v případě elektronických součástek, které jsou základním kamenem dnes již většiny zařízení? Návrh spolehlivého systému je věcí poměrně složitou, a tak stále ještě výsledná spolehlivost záleží hlavně na financích.

Spolehlivost spolu s odolností proti poruchám (Fault Tolerant – FT) tvoří významnou kapitolu při navrhování číslicových obvodů. V praxi se běžně používá pro zlepšení spolehlivosti systému replikace funkčních jednotek a následné vyhodnocování paralelně zpracovaných výstupů. Zde se však potýkáme se zvýšenými nároky na zdroj obvodu a zvýšenou spotřebou energie, což nemusí být vždy přijatelný faktor. Pokud vyvíjíme systém odolný proti poruchám, setkáváme se dále s vyššími nároky na návrh řízení a testování obvodu, proto je vhodné stále zkoumat další metody testování obvodu a metody pro návrh systémů odolných proti poruchám [10].

V rámci výzkumných aktivit na Vysokém učení technickém v Brně byla navržena kompletní metodologie pro tvorbu systémů se zvýšenou spolehlivostí využívající techniky odolnosti proti poruchám založené na bázi obvodů FPGA. Pro zvyšování odolnosti systému proti poruchám se doposud nejčastěji využívaly techniky TMR a duplex nebo bezpečnostní kódy [4]. Hlídací obvody mezi těmito technikami nenašly příliš velké použití. To se však mění s návrhem nové metodologie, která se opírá o využití hlídacích obvodů v různých architekturách odolných proti poruchám. Nejnižší vrstvu výzkumu tvorby FT systémů tvoří metodologie založená na automatizovaném vytváření hlídacích obvodů pro testování správnosti komunikačních protokolů [11].

Hlavním cílem je vyvinout portál, který nabídne možnost jednoduchého vytváření projektů za pomoci již přednastavených bloků s libovolným nástrojem. Výhodou by pak měla být jednoduchost zpracování, kdy uživatel zadá pouze vstupní soubory a vybere bloky,

které mají tyto soubory zpracovávat. Motivací této práce je, aby výsledek projektu byl vždy úplný a uživatel již nemusel hledat další nástroj na následné zpracování.

Výhodou takového komplexního webového portálu pak je nejen rychlý a univerzální přístup k projektu, ale také možnost vytvoření projektu bez hlubších znalostí problematiky popsané v dalších kapitolách. Následně také možnost efektivně používat a sdílet vytvořené nástroje, které tyto problematiky řeší.

Námět pro vznik tohoto portálu je popsán v druhé kapitole této práce. Třetí kapitola je věnována architekturám odolným proti poruchám a kapitola čtvrtá popisuje analýzu a specifikaci celého systému. Následný návrh systému je uveden v kapitole páté, na kterou volně navazuje kapitola šestá, představující průběh a problémy implementace. V sedmé kapitole jsou zveřejněny parametry testování systému společně s instrukcemi pro práci s implementovaným systémem. Závěr, tvořící osmou a závěrečnou kapitolu shrnuje získané poznatky a je taktéž zaměřen na pohled dalšího vývoje směrem k diplomové práci.

Kapitola 2

Námět pro vznik portálu

V předchozí kapitole byly uvedeny širší souvislosti práce v reálném světě. V této kapitole se zaměříme na důvody, které podněcují k tvorbě tohoto systému. Vezmeme v úvahu dosavadní dostupná řešení a podíváme se na způsoby, kterými se lze v dalších kapitolách ubírat.

2.1 Současné možnosti vývoje

Představme si situaci, kdy máme k dispozici popis architektury v libovolném formátu a mnoho nástrojů od různých autorů pro převod či modifikace této architektury. Tyto nástroje se postupem času razantně mění a pokaždé, když se objeví nová technologie, či se nepatrně změní postupy, je nástroj výrazně upraven, či nahrazen nějakým novým s naprosto jiným vstupním a výstupním rozhraním. Pokud máme sadu nástrojů od jediného autora, můžeme předpokládat, že vstupní a výstupní formáty budou stejné a není problém popis naší architektury přizpůsobit požadavkům nástrojů. Ovšem pokud budeme chtít používat nástroje od různých autorů, jen těžko je všechny donutíme, aby na vstupu očekávali námi zadaný formát.

V této situaci nezbývá nic jiného, než pro každý z používaných nástrojů vytvořit i konvertor vstupního či výstupního formátu. A tak vzniká další práce navíc. Nehledě na to, že pokud budeme používat více samostatných nástrojů, pokaždé musíme jejich výstup přivést na vstup dalšího nástroje. Tento problém se dá řešit mnoha způsoby. Jedním z nich je napsat pro konkrétní příklad také skript, který všechnu práci obstará za nás. Ovšem každá změna v nástroji vyžaduje novou kontrolu celého takového systému.

Dalším problémem současného řešení je distribuce vývojových nástrojů. Každý nástroj může být silně specifický a návrháři takového nástroje nic nebrání si vymyslet vlastní popis vstupu či výstupu. Tento efekt má za následek to, že pokud budeme chtít používat cizí nástroj, pro který není připraveno univerzální rozhraní, budeme muset vše vytvořit my. To je ovšem časově náročné, a pokud budeme chtít náš nástroj rozšířit mezi ostatní uživatele, musíme být schopni jim to nějak ulehčit.

2.2 Možnosti změny

Při zamyšlení nad danou situací přicházíme k poznatku, že by bylo vhodné každý nástroj zabalit do takzvané *černé skříňky*, která by tvořila samostatný krok v celém projektu. Tato *černá skříňka*, tvořící „krok“ projektu by pak mohla mít jednotný vstup i výstup a ve výsledném systému by se s ní dalo pracovat jako s jakýmkoli jiným nástrojem. Z takto

definovaných *kroků* by pak mohl být složen celý projekt a uživatel by již pouze pracoval v rámci tohoto projektu, který by již bylo snadné vizualizovat na ploše. Ztratily by se tedy problémy s těžkou orientací v mnoha navzájem propojených krocích a předávání souborů z jednoho kroku do druhého by mohl obstarávat program.

Tohoto cíle lze dosáhnout, pokud definici jediného kroku rozdělíme na sled několika základních úkolů. Prvním z nich bude vložení spustitelného nástroje do systému, případně hned několika nástrojů, na sobě přímo závislých. Závěrečným úkolem by pak měla být identifikace důležitých parametrů, které tento nový nástroj očekává na svém vstupu, případně jakého typu tyto parametry jsou. Takto definovaný krok by měl být nyní hotov a připraven k použití již jako ona *černá skříňka*. Pokud budeme mít tento definovaný krok, pak jej můžeme spojit s libovolným dalším krokem a tím propojit celou sadu nástrojů v rámci uživatelova projektu.

Takto definované kroky je možné jednoduše sdílet s ostatními uživateli, kteří mohou schéma propojení kroků definovat odlišně a tím lépe postihnout potřeby jejich projektů. Pokud tedy navrhneme takovýto systém, pak je potřeba rovněž rozmyslet možnosti jeho přenositelnosti. Hlavní výhodou by se měla stát abstrakce jednotlivých nástrojů a následné jednoduché sdílení těchto postupů. Nabízí se tedy možnost spojit tyto výhody s výhodami webového portálu a umožnit přístup k tomuto systému kdekoli na základě přístupových práv rozdílných skupin uživatelů.

Kapitola 3

Architektury odolné proti poruchám

V předchozí kapitole jsme si obecně popsali problémy současného řešení práce s architekturami odolných proti poruchám. V této kapitole se tedy podrobněji zaměříme na teorii, která je důležitá pro pochopení funkčnosti programů, které budou v základu obsluhovány. Uvedeme metody, které zajišťují rezistenci proti chybám, jež se mohou v navržených architekturách vyskytovat. Taktéž se obeznámíme i s dostupnými nástroji pro generování hlídacích obvodů v architekturách odolných proti poruchám, tzv. Fault Tolerant systémech. Tyto nástroje budou později tvořit programy, které poslouží jako vhodné jádro „černých skříněk“, se kterými bude ve výsledku uživatel přicházet do styku.

3.1 Metody pro zajištění odolnosti architektury

Termín spolehlivost se zabývá pravděpodobností, zda se systém v daném čase porouchá, což znamená, že výstupy systému nebudou správné. Avšak poruchy, které výstupy negativně ovlivňují, jsou v elektrotechnice velmi častým jevem. Při navrhování odolné architektury je potřeba s nimi počítat.

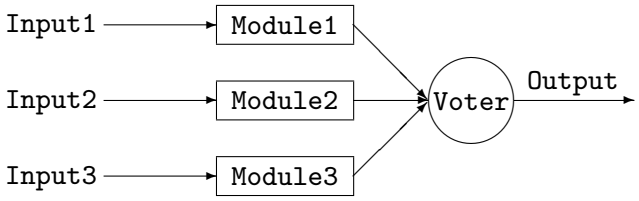
Při nahrazování mechanických jednotek elektronickými (rozdělovač, regulátor, apod.) došlo k výraznému snížení spolehlivosti systému. Pokud ale chceme spolehlivost zachovat, je třeba vytvářet systémy, které mohou dále provádět zadanou úlohu za přítomnosti obvodových poruch a programových chyb. K tomu existují následující tři základní metodiky, které pomáhají eliminovat vliv chyby na systém: předcházení chybám (fault avoidance), maskování chyb (fault masking) a eliminování vlivu chyby (fault tolerance). Tyto metodiky zahrnují několik technik pro konkrétní zpracování eliminace chyby, kterým je potřeba se věnovat právě v místech, kde je jakákoli chyba nepřípustná. Příkladem může být vesmírný a letecký průmysl, železniční signalizační systém, medicínské a život udržující systémy či nukleární a chemické procesy. Pokud se v těchto systémech chyba vyskytne, je potřeba, aby ji řídicí systém zaregistroval, případně ji dovedl odstranit.

Každá technika pro eliminaci chyb skýtá nevýhodu v nadbytečné replikaci komponent, a následně navýšené spotřeby prostoru, energie či materiálu. Záložní systém vždy funguje jako náhrada za systém, ve kterém se vyskytla chyba. Při určování výsledné spolehlivosti nově sestaveného systému záleží na každé technice, jak záložní systém vyhodnocuje ve vztahu se systémem původním, stejně tak, jako na počtu záložních systémů. Je tedy nutné používat techniky ohleduplně a testovat, zda třeba i úspornější technika neplní stejný účel.

K zajištění spolehlivosti systému se nejčastěji používají techniky TMR a duplex [7].

3.1.1 TMR

Systém TMR je zkratkou z anglického Triple Modular Redundancy. Tento systém patří do skupiny systémů M *z* N, ve kterých je nutné ke správné činnosti systému jejich M prvků z celkových N prvků. Pod termínem TMR se rozumí paralelní zapojení tří prvků tak, aby výpadek jednoho vedl k maskování poruchy v systému.



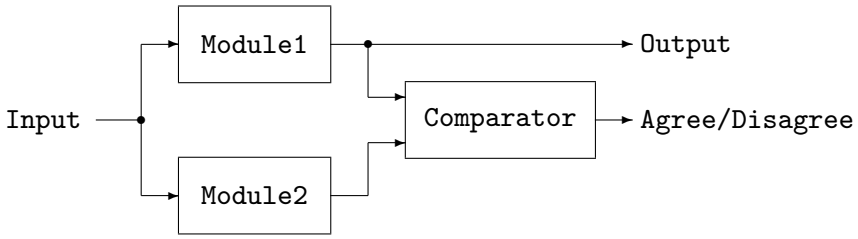
Obrázek 3.1: Schéma systému TMR.

Na obrázku 3.1, je zobrazen výstup od každé komponenty, veden na vstup rozhodovacího členu (Voter). Pokud se na jednom z členů vyskytne chybný výstup, rozhodovací člen jej sečte s ostatními nechybujícími výstupy a chyba se na výsledném výstupu neprojeví v plné míře. Systém může pracovat pouze s jedním chybným prvkem. V tomto systému se očekává od každého členu stejný výstup a při případném posouzení se zbylými prvky se navíc odhalí chybující prvek, který je možné následně rekonfigurovat.

TMR je běžně používáno ve vesmírné letecké elektrotechnice [2].

3.1.2 Duplex

Jedná se o techniku, využívající zdvojení komponent se srovnávacím členem na výstupu. Obrázek 3.2 popisuje strukturu zapojení. Oproti technikám M *z* N je zaveden chybový výstup, určující, zda porovnávané komponenty mají stejný, nebo rozdílný výstup. Pokud je zjištěn rozdílný chod, nebo případná porucha jedné z komponent, bude na chybový výstup odeslána informace o chybě. Z tohoto vyplývá, že systém je při výskytu chyby nefunkční.



Obrázek 3.2: Schéma techniky zdvojení se srovnávacím členem.

Zdvojení komponent zvyšuje bezpečnost zařízení a současně snižuje jeho spolehlivost. Tyto dva parametry mohou působit protichůdně, ale ve výsledku záleží na tom, pro jaké účely bude tato technika použita. V praxi použití této metody znamená zvýšení pravděpodobnosti nahlášení „planého poplachu“ za cenu vyšší bezpečnosti.

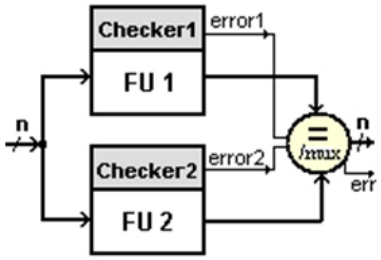
3.2 Hlídací obvody

Hlavní použití těchto prvků spočívá v kontrole komunikačních protokolů.

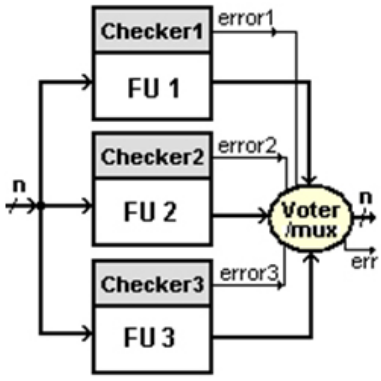
Hlídací obvod je založen na principu přídavného obvodu s obdobným rozhraním jako hlídaná komponenta. Vstupy i výstupy zůstávají zachovány, navíc je zaveden pouze chybový výstup. Výhodou tohoto výstupu je, že v případě výskytu chyby označí komponentu a je možné provést pouze částečnou rekonfiguraci v místě výskytu chyby. Hlídací obvody mohou být realizovány buďto jako vnitřní obvod, kde se pouze změní rozhraní přidáním chybového výstupu, nebo vnější obvod, který replikuje rozhraní původního a posuzuje chování podle předem stanoveného protokolu.

3.3 Aplikace hlídacích obvodů

Největší síla využití hlídacích obvodů je ve sloučení s jinou často používanou architekturou. V kombinaci s architekturami TMR nebo duplex vznikne mnohem odolnější systém proti chybám. Hlídač má schopnost detekovat a lokalizovat poruchu modulu. Klasický systém duplex v případě jednoho chybného prvku není schopen pracovat. Pokud je každý z prvků opatřen hlídacím obvodem jako na obrázku 3.3, architektura poskytuje správné výsledky i při poruše jednoho modulu. Taktéž se rozšíří spolehlivost systému TMR stejným použitím. Při sestavení architektury podle obrázku 3.4 vznikne systém, který je schopen vracet správné výsledky i při poruše dvou modulů.



Obrázek 3.3: Duplex architektura s využitím hlídacích obvodů.



Obrázek 3.4: TMR architektura s využitím hlídacích obvodů.

Hlídací obvod lze využít pro detekci poruchy v hlídané komponentě a k její lokalizaci. Tato funkce určuje hlavní směr použití hlídacích obvodů k tvorbě FT systémů. Mimo jiné

lze hlídací obvody použít pro verifikaci návrhu či k průběžné diagnostice.

3.4 Dostupné nástroje pro tvorbu FT systému

Pro účely webového portálu jsou využívány tři nástroje, které vznikly na UPSY — FIT VUT v Brně. Slouží jako názorná ukázka programů, se kterými se bude v systému pracovat, a navíc plně implementují metodiky pro zvyšování spolehlivosti.

Prvním programem je generátor hlídacích obvodů. Program je napsán v jazyce C a za vstup přijímá soubor s formální definicí komunikačního protokolu, nebo číslicového obvodu, pro kterou následně vytvoří hlídací obvod. Výstupem programu je již číslicový obvod v jazyce VHDL.

Dalším nástrojem vhodným pro účely portálu je program pro tvorbu TMR a duplexních architektur HDL. Tento program, taktéž napsaný v jazyce C, generuje několik nových obvodů z jednoho vstupního popisu obvodu. Zadáním parametru můžeme ovlivnit, zda se vygeneruje pět nových souborů pro systém TMR (tři kopie původní komponenty, rozhodovací člen a zastřešující komponenta), nebo čtyři nové soubory pro duplexní systém (dvě kopie původní komponenty, porovnávací člen a zastřešující komponenta).

Posledním nástrojem, který je nápomocen uživateli při tvorbě FT systému, je program — napsaný v jazyce C — pro převod VHDL kódu na formální popis architektury. Za vstup přijímá VHDL kód zadané architektury a výstup je nově vytvořený soubor s příponou `.frm`, obsahující formální popis zadané architektury.

Všechny zmíněné programy jsou založené na práci s VHDL kódy, popisujícími pouze komunikační protokoly nebo číslicové obvody popsané na úrovni meziregistrových přenosů RTL. Pro jiné VHDL kódy nebude generování fungovat správně.

Kapitola 4

Analýza a specifikace požadovaného systému

V předchozí kapitole byla shrnuta teorie, potřebná k pochopení problematiky související s tímto projektem. V návaznosti na tuto teorii jsme schopni shrnout a analyzovat požadavky, které budou mít uživatelé pracující s touto aplikací.

Nejprve se rozbor zaměřuje na účel aplikace a očekávání ze strany požadavků na systém. Důležitou součástí aplikace je pak i zobrazování bloků v rámci aplikace, a proto jsou analyzovány i možnost tohoto zobrazování v rámci webové aplikace. Vzhledem k povaze systému, který je schopný obstarávat spustitelné soubory je pak nezanedbatelnou součástí tématu analýzy také otázka bezpečnosti výsledného systému

Všechny tyto části mají za úkol podrobně rozebrat a specifikovat problémy a situace, jejichž řešení jsou následně navržena v další kapitole.

4.1 Účel aplikace

Z předchozí kapitoly máme povědomí o nástrojích, které je možné obsluhovat. Cílem implementační části této práce je vytvořit webovou aplikaci, která pod sebou bude tyto a další obdobné programy sdružovat a poskytne k nim vhodné pracovní prostředí. Nyní si tedy abstraktně popíšeme všechna očekávání od této aplikace.

4.1.1 Shromažďování spustitelných souborů

Opěrným bodem celého systému jsou programy, čekající na zařazení do uživatelova projektu. Tyto programy jsou již kompilované a spustitelné na serverovém stroji. Jejich tvorbu a ověření funkčnosti má na starosti osoba poučená a není potřeba tuto věc již v rámci systému řešit. Tyto programy je potřeba systémem pouze nahrávat na stanici společně s možností jejich spuštění. Přístup koncového uživatele k těmto blokům by proto měl být omezen pouze na povolené způsoby vykonání programu. Očekávané možnosti spuštění programu jsou tedy pomocí zadaných parametrů pro spuštění. Každý program může mít celou sadu těchto parametrů a v systému se očekává vedení těchto parametrů přímo vázaných na spustitelný program. Pomocí těchto parametrů se programu může předávat pouze hodnota. Tato hodnota však může odkazovat na cestu k souboru, který program očekává jako svůj vstup, případně cestu ke složce, kterou zpracovává.

Programy je tedy potřeba shromažďovat v abstraktním bloku, který uživatel nebude moci měnit jinak, než přes systémem nabízené parametry. V rámci nahrávání tohoto abs-

traktního bloku by měla být možnost nahrání více vzájemně spolupracujících programů. Některé programy mohou potřebovat ke svému běhu soubor, který vygeneruje jiný, spuštěný a úspěšně dokončený v čase před ním. Sled těchto programů zná osoba, která nahrává programy do systému předem a jejich kooperaci má již ověřenou. Stejně tak je potřebné, aby byla zavedena volitelná možnost, kdy vybraný soubor, či více souborů může být prováděno paralelně s ostatními, sériově prováděnými, programy. Serverová stanice může obsluhovat více procesorů, a proto je třeba, aby i s tímto navržený systém počítal a nevynucoval pouze sériové provádění programů. Při vkládání každého programu do systému definujeme pak pro každý samostatný program sérii vlastních parametrů pro spuštění a u nich označíme ty, které je v rámci projektu možné editovat a doplnit v nich vlastní hodnotu, případně pomocí nich odkázat na výstup jiného bloku v rámci vytvářeného projektu.

Pro označení této skupiny spustitelných programů, která má definovány potřebné parametry a vstupy nadále označujme jako „*blok*“ a z pohledu uživatele tento blok chápejme jako předem již definovaný komplexní program, který je možné vložit do své vlastní práce, zvané „*projekt*“. V rámci projektu je možné tento *blok* také označit za jeden *krok* projektu — tyto výrazy jsou synonymní a liší se pouze v pohledu na tu samou skupinu zabalených programů společně s parametry ze strany správce, který jej do systému zadal a ze strany uživatele, který tuto skupinu používá jako celek v rámci již větší skupiny.

Vhodnou možností pro ulehčení tvorby těchto bloků je možnost klonování již existujících, což znamená, že pokud již byl nějaký blok vytvořen a uživatel potřebuje zavést podobný blok pouze odebráním či přidáním nějakého dílčího programu, nebo pouze potřebuje jinak definovat jeho vnitřní parametry, pak pouze zvolí možnost blok klonovat a systém sám zkopíruje všechna potřebná data a spustitelné programy a s jejich pomocí vytvoří zcela nový blok, který lze jednoduše opravit stejně, jako při vytváření či editaci stávajícího bloku. Stejně tak je potřeba určit, zda již je blok zcela dokončen a vytvořen a může již být poskytnut k tvorbě projektu, který jej bude zahrnovat. Takovýto blok si může uživatel vytvořit také pouze sám pro sebe a nemusí jej poskytnout ke zveřejnění ostatním uživatelům, je však potřeba tuto možnost odlišovat od výše zmíněné možnosti blok označit za použitelný v projektu. Definujme tedy dva pojmy: *použitelný blok* a *publikovaný krok*, kde první z termínů znamená, že krok již lze zahrnout do projektu a druhý termín označuje skutečnost, že pokud je blok označen jako *použitelný* a zároveň *publikovaný*, pak jej může do svého projektu vložit i kdokoli jiný, než pouze jeho autor. Bloky si tedy nesou označení svého majitele a uživatel může vidět, kdo je autorem bloku, který se chystá použít.

4.1.2 Přístup k programům

Zavedení programů do systému je pouze jednou částí systému. Tou druhou je následné zavedení přístupu k těmto blokům, které se bude dít v rámci projektů. Uživateli je potřeba nabídnout jednoduché pracovní prostředí, kde bude skladovat vlastní vytvořené projekty. Každý projekt má svého vlastníka, který tento projekt vytvořil a také stejně jako v definici bloku má i projekt možnost stát se veřejně dostupným. Takovýto projekt je pak ve výpisu projektů oddělen od projektů vlastních a je u něj zobrazena informace, který uživatel je jeho autorem. Tento veřejný projekt ovšem není volně přístupný k editaci, ale pouze ho lze klonovat a tak z něho vytvořit projekt vlastní — veřejný projekt tedy každý jiný uživatel než jeho tvůrce vidí jako pouhou *šablonu* pro vytvoření svého vlastního projektu.

Nyní se zaměříme na to, jak vypadá a co zahrnuje vlastní tvorba projektu. Projektem v aplikaci chápeme množinu předdefinovaných bloků, které v aplikaci tvoří jednotlivé proveditelné kroky, které mezi sebou mohou být provázány podle uživatelem vytvořených

závislostí. Tuto definici bude nutné blíže specifikovat následujícím popisem. Pokud vložený krok uvnitř své definice obsahuje editovatelné parametry, pak po vstoupení na tento prvek může uživatel změnit hodnotu daného parametru, což může být hodnota, ale také soubor, který generuje jiný krok projektu, čímž vnikne vazba na jiný krok a tento krok musí být bezpodmínečně spuštěn před krokem, který soubor očekává na svém vstupu — kroky, které nejsou svázány, mohou být spouštěny paralelně. Čímž se již dostáváme k výsledné části tvorby projektu, kterou je spouštění projektu.

Projekt po dokončení úprav potřebných bloků může být buďto odložen k pozdější editaci, či bez jediného spuštění může sloužit pouze jako veřejná šablona pro tvorbu projektů z něho vycházejících, ale hlavní potřebou systému je možnost projekt spustit a vykonat všechny jeho kroky. Ve vytvořeném projektu se tedy musí najít správná posloupnost kroků, které na sebe musejí čekat a ve správném pořadí všechny programy obsažené v jednotlivých krocích spustit. Webová strana aplikace však nesmí na dokončení projektu čekat, neboť zpracování celého programu může být libovolně dlouhé a mezitím může uživatel předat ke zpracování další projekt, případně pokračovat v další tvorbě bez čekání. Je tedy hlavní potřeba zpracování projektu zcela oddělit od části webových stránek, neboť se předpokládá, že uživatel se během zpracování může od systému zcela odhlásit a vypnout webový prohlížeč.

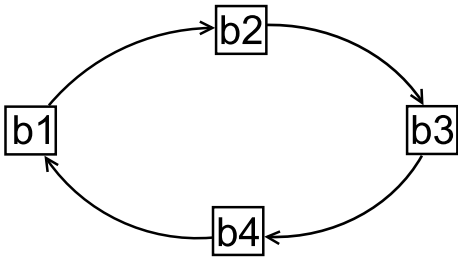
Po celkovém vykonání programu je potřeba výsledek projektu vhodně zobrazit a nabídnout uživateli stažení vygenerovaných souborů, pokud běh skončil bez chyb. V případě běhu s chybou je nutné, aby uživatel zjistil, který blok chybu provedl a mohl případný blok z projektu vyjmout, či změnit parametry, které k chybě vedly.

4.1.3 Zobrazení projektu

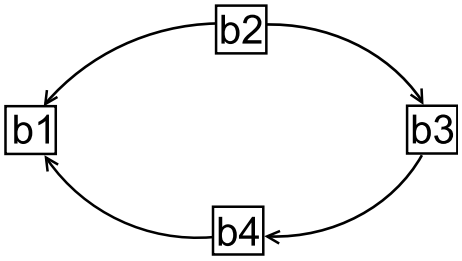
Hlavní část tvorby projektu tvoří jeho vlastní vyobrazení. Projekt je tvořen pojmenovanými bloky, které lze mezi sebou propojovat. Pokud bychom zvolili cestu textové reprezentace tohoto schématu, stala by se tvorba projektu věcí značně nepřehlednou již při malém počtu bloků. Musíme si představit situaci, kdy uživatel zadává spojení mezi vícero bloky, kde do jediného bloku může vést hned několik spojení z různých jiných bloků. Pokud bychom vypisovaly bloky pouze v rámci seznamu, i interaktivní oddělení spojení, kdy jsou spojení vyznačena po najetí kurzorem na žádaný blok, či na jediné spojení se může lehce stát matoucím a neintuitivním, nehledě na složitou implementaci pouze pomocí standardních HTML prvků. Výrazně lepší reprezentaci tedy nabízí možnost grafické reprezentace vložených dat. Pokud vytvoříme ze zadaných bloků a jejich propojení úhledný planární graf, pak uživateli nabídneme možnost rychlé orientace a pohodlného pracovního prostředí, kdy nemusí spojení dále dohledávat, ale stačí spojení pouze definovat pro jednotlivý krok, což se ve výsledku promítne do grafické reprezentace.

Můžeme tedy říct, že se bude jednat o blokové schéma orientovaného grafu. Tento graf bude mít jako uzly bloky a jako hrany mu poslouží právě spojení, které zajišťuje fakt, že některý z programů v bloku očekává v rámci svého parametru soubor, který je generován jiným blokem. Celý projekt může být tvořen jako nesouvislý orientovaný graf, který nemá další obecná omezení při tvorbě grafu. Ovšem pro spuštění projektu je nutné, aby v grafu neexistovala kružnice, kde všechny orientované hrany směřují jedním směrem v dané kružnici — tato situace by znamenala, že se v projektu vyskytuje blok, který jako vstup očekává soubor, který generuje blok provedený až po čekajícím bloku, neboli by nastalo *uváznutí* (angl. Deadlock), což je nepřípustná situace. V této situaci by se eventuálně projekt mohl vykonat, neboť některé programy by ohlásily chybu, ovšem tato situace není vždy zajištěna a proto je nutné tento stav detekovat již předem a v případě výskytu kružnice,

vedoucí k uváznutí spuštění projektu nepovolit. Situaci vedoucí k uváznutí můžeme vidět na obrázku 4.1, zatímco na obrázku 4.2 vedou do uzlu `b1` hrany z uzlů `b2` a `b4`, což je již korektní situace. Jak je z obrázků patrné, obě situace obsahují kružnici v orientovaném grafu, ovšem s důležitým rozdílem, kterým je právě shodná orientace všem hran jedním směrem po kružnici (obrázek 4.1).



Obrázek 4.1: Výskyt kružnice, vedoucí k uváznutí.



Obrázek 4.2: Korektní situace nevedoucí k uváznutí.

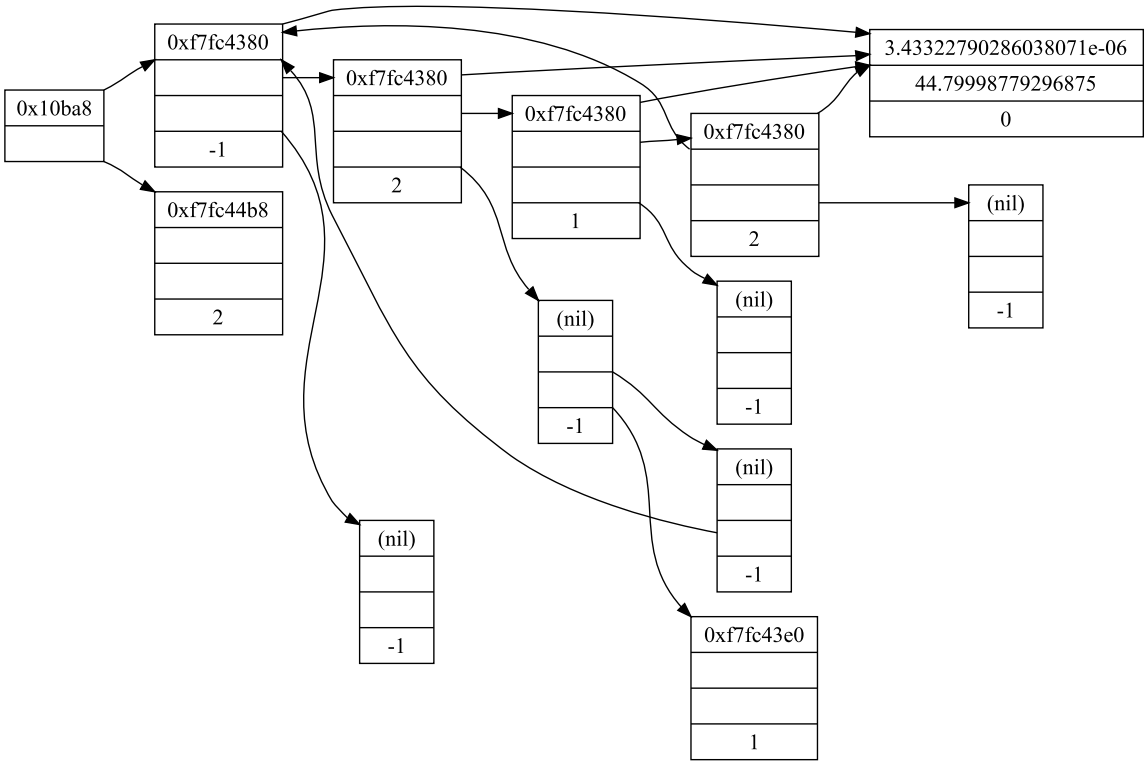
V grafu se mohou vyskytnout zcela oddělené podgrafy, které pro své provedení nepotřebují žádný cizí vstup a je tedy vhodné je spouštět paralelně s ostatními podgrafy, ve kterých musí být nalezena posloupnost, ve které lze tyto bloky spouštět.

4.2 Možnosti zobrazování bloků

Vhledem k vlastnostem grafického výstupu projektu zmíněných v předchozích odstavcích a grafu, který zobrazuje je jasné, že potřebujeme nástroj, který podle zadaných vlastností tento výstup bude umět sám vyprodukovat. Tento program musí být dostatečně rychlý, aby bylo možné jej spustit během samotného generování stránky a byl srovnatelně rychlý s načtením obrázku uloženého na disku. Jednou z možností je generovat obrázek přímo v kódu, který se stará o tvorbu samotné stránky, která se zobrazí uživateli, což je v tomto případě jazyk PHP. Ovšem generování schématu obnáší specializované algoritmy pro hledání rozložení uzlů společně s rozvržením hran, tak, aby byl graf planární. V tomto oboru již existuje více nástrojů, které dané téma zpracovávají, a proto je tedy vhodnější sáhnout po již hotovém nástroji, který je možné pouze externě volat a zobrazovat jeho výstupy. Další podmínkou, kterou musí hledaný program splňovat je tedy formát výstupu. S grafickým výstupem je potřeba nadále ještě pracovat a mít možnost vložit do tohoto grafického výstupu odkaz pro volání konkrétního bloku ze schématu. Možné formáty, které tento požadavek splňují, jsou tedy textové formáty pro popis grafického výstupu, ve kterých lze zpětně dohledat zobrazované prvky. Pro použití v moderní webové aplikaci je proto ideální zvolit

formát SVG (Scalable Vector Graphics), což je značkovací jazyk a formát souboru popisující dvojrozměrnou vektorovou grafiku pomocí XML [15]. Tento formát je plně podporován všemi prohlížeči v aktuální verzi.

Těmto zavedeným podmínkám pro hledaný program plně vyhovuje program *Graphviz*, který umožňuje vizualizaci grafů pomocí strukturovaného popisu vkládaných informací [6]. Navíc v něm existuje přímo vestavěná možnost vložit do popisu dat odkaz, na který vykreslený prvek bude odkazovat, čímž výrazně usnadňuje následnou práci a zpracování obrázku. V tomto programu si lze jednoduše definovat celou strukturu požadovaného grafu a poté tyto instrukce předložit dílčímu programu `dot` z komplexu programů pro vizualizaci grafů a sítí *Graphviz*. Vygenerování výsledné vektorové grafiky ve formátu `.svg` je srovnatelně rychlé s generováním jednoduchého obrázku v rámci jazyka PHP a proto je tedy vhodné použití tohoto externího programu. Program `dot` byl pak vybrán pro vhodnost grafického výstupu, který vizuálně přesně odpovídá potřebám pro vykreslení blokového schématu. Pro ilustraci nechť poslouží obrázek 4.3, který vyobrazuje ukázkový příklad možného rozvržení datové struktury popsaného vlastním jazykem pro definici vypisované struktury.



Obrázek 4.3: Ukázkový výstup programu `dot`.

4.3 Bezpečnost systému

Nedílnou součástí každého webového portálu musí být téma jeho zabezpečení z hlediska možnosti vnějších útoků. Každý veřejně dostupný webový portál může být napadnut mnoha různými způsoby a proto je potřeba předem vědět o největších slabinách systému, abychom tyto skutečnosti mohli lépe ošetřit a vyhnout se případným problémům se ztrátou dat, či poškození serverové stanice.

4.3.1 Spustitelné soubory na serveru

Tento webový portál musí disponovat možností nahrát na server přeložený spustitelný soubor a nastavit u něho oprávnění k jeho spuštění. Tato situace sama o sobě je nebezpečná hned z několika důvodů. Nahraný program v sobě může nést spoustu chyb, které při jeho spuštění vnese do systému. Těmito chybami může být to, že program za běhu může mazat, či měnit důležitá data, nebo může úmyslně vytěžovat procesor a tak ho blokovat ostatním procesům. Pokud by měl přístup k nahrávání programů na server kdokoli, je velká pravděpodobnost, že se server dříve či později stane terčem útoků, či pokusů o ně. Z této situace vyplývá, že možnost nahrávání spustitelných souborů na server přes webové rozhraní je velmi nebezpečné a je třeba tuto situaci dostatečně zabezpečit.

Pokud uvažujeme implementační jazyk PHP, ve kterém je aplikace tvořena, pak víme o možnostech zabezpečení, které PHP server standardně implementuje a lze těchto mechanizmů využít. Ovšem tyto mechanismy jsou určeny k tomu, aby standardně všem typům nebezpečných situací bránily hned v počátcích, a nedovolí vzniku potenciálně nebezpečných situací. Typickými mechanismy může být zakázání běhu potenciálně nebezpečných funkcí, jakými jsou například `chmod()` a `chown()`, které mění oprávnění přístupu k souboru, nebo `unlink()`, `rename()` či `copy()`, které manipulují přímo se soubory a mnoho dalších funkcí, které stejně jako již vyjmenované zakazuje tzv. `save_mode`, jenž lze nastavit v inicializačním souboru pro spuštění PHP. Tyto funkce jsou však důležité pro manipulaci se soubory na serveru, stejně tak pro možnost vykonání spustitelného souboru ze strany webové aplikace.

Z předchozího rozboru tedy vyplývá, že musíme bezpečnost řešit jinými způsoby, než standardními preventivními, které nabízí samotné PHP. Jako jeden ze způsobů řešení se nabízí možnost povolení přístupu k systému pouze vybraným osobám, které budou moci na server potřebné soubory nahrávat až v případě, kdy vědí, že jsou bezpečné a dostatečně ověřené. Druhou možností je pak webovou aplikaci provozovat pouze v lokální síti a povolit k ní přístup jen autorizovaným osobám, které již mají přístup do celého systému. Ovšem toto rozhodnutí by mělo být již na provozovateli webového portálu a v navrženém systému je potřeba implementovat rozdělení rolí a s nimi související oprávnění k nahrávání spustitelných souborů a možnosti jejich spouštění. Předpoklad webového portálu tedy je, že s ním budou pracovat poučení uživatelé, kteří mají možnost přímého přístupu na server, a tedy není potřeba bránit jim v nahrávání souborů přes webovou aplikaci. Stejně tak ovšem k systému mohou přistupovat i uživatelé bez oprávněného přístupu na server, pro které již musí být práva omezena a pro ně je předpokladem pouze možnost spouštět již nahrané soubory.

4.4 Uživatelské rozhraní

V předchozí sekci jsme v závěru nastínili příklad možných uživatelů, kteří budou portál využívat. Nynější sekce má pak za účel toto téma rozvinout a věnovat se možnosti vystavění rozhraní, které bude vyhovovat právě potřebám cílových uživatelů a umožní jim efektivní a pohodlnou práci se systémem.

4.4.1 Obsluha a návštěvníci portálu

Při zavedení tohoto portálu do provozu uvažujeme dva odlišné typy uživatelů pracujících se systémem. Prvním typem je obsluha portálu, která vlastní přístupová práva k serveru a má i bez systému možnost spouštět a nahrávat soubory na server. Tito uživatelé jsou dobře obeznámeni s programy, které obsluhují a portál pro ně přináší možnost mít přehlednou

správu nad těmito programy a poskytnout k nim přístup i ostatním. Jsou to taktéž ve většině případů uživatelé zkušení, kteří mají v oblasti informačních technologií velmi pokročilé znalosti a jsou zvyklí na práci s mnohými složitými informačními systémy. Pro ulehčení práce této skupině je vhodné se držet zaběhlých způsobů rozložení a umístění ovládacích prvků a zjednodušit ovládání nejčastěji používaných věcí, neboť jejich spokojenost závisí především na časové úspoře při práci se systémem. Taktéž tato skupina bude tou, která se bude snažit systému porozumět celkově, aby mohla školit druhou skupinu uživatelů, kteří budou pracovat se systémem po zavedení dat (především po definování spustitelných souborů) ze strany první skupiny.

Druhou skupinu pak tedy tvoří uživatelé, od kterých nemůžeme čekat tak rozsáhlé znalosti systému a stejně tak musíme počítat i s menšími uživatelskými schopnostmi, neboť to mohou být i začátečníci, kterým tento portál pomáhá s výukou jiné problematiky, využívající výhod a schopností tohoto portálu. Obě skupiny mají shodnou potřebu jednoduchého systému, který jim nabídne rychlou orientaci bez dlouhého školení, které může proběhnout v rámci čtení dokumentace a není tedy potřeba jej opakovat v rámci uživatelského rozhraní. To by se mělo zdržet zbytečných textů, které přímo nesouvisí s popisem ovládacích prvků.

Návrh aplikace by se tedy měl držet jednoduchého stylu, ze kterého lze jasně rozpoznat důležité ovládací prvky a intuitivně lze odtušit, co je úkolem v dané obrazovce a jak přejít k dalšímu potřebnému úkolu. Stejně tak se musí rozvržení úkolů držet dobře strukturovaného schématu a uživatel nesmí být nucen procházet mnoho obrazovek s drobnými informacemi, aby se dostal k věci, kterou opravdu hledá, či potřebuje vykonat. Portál by tedy měl být zbaven zbytečných popisů a funkcí, které lze rozhodnout bez účasti uživatele, či lze předpokládat. Systém by taktéž měl využívat moderní prvky ulehčující práci a využívat vhodné prvky pro práci s myší, místo práce s klávesnicí podobně jako by měl preferovat možnost přetažení souborů přímo do pracovní plochy aplikace místo vyhledání souboru ve složce.

Webový portál je směřován pro použití v aktuálních verzích prohlížečů, ovšem funkčnost důležitých prvků musí být zachována i na starších prohlížečích, nehledě na systém. Při omezení verzí prohlížeče smí být vynechány pouze prvky zjednodušující použití systému a mohou být nahrazeny funkčními, méně efektivními prvky.

Kapitola 5

Návrh systému

V předchozí kapitole byly analyzovány všechny hlavní problémy, kterými se musí zabývat návrh systému. Nastíněny byly i problémy zobrazování projektu a problém bezpečnosti.

V této kapitole detailně popíšeme diagram případů užití, který názorně ilustruje všechny části systému, na které se návrh zaměřuje, a které jsou v implementační části diplomové práce zpracovány. Dále se přesuneme k detailnějšímu řešení problémů z předchozí kapitoly a několika dalších, rozdělíme aplikaci do dvou hlavních celků a toto provedení odůvodníme. Budeme se věnovat popisu komunikace těchto částí a u každé části podrobně rozvrhneme, jaké pole působnosti spadá pod její pravomoci. Tato část je klíčovým bodem, ze kterého vychází implementované řešení a kde jsou uvedena všechna reálná řešení systému.

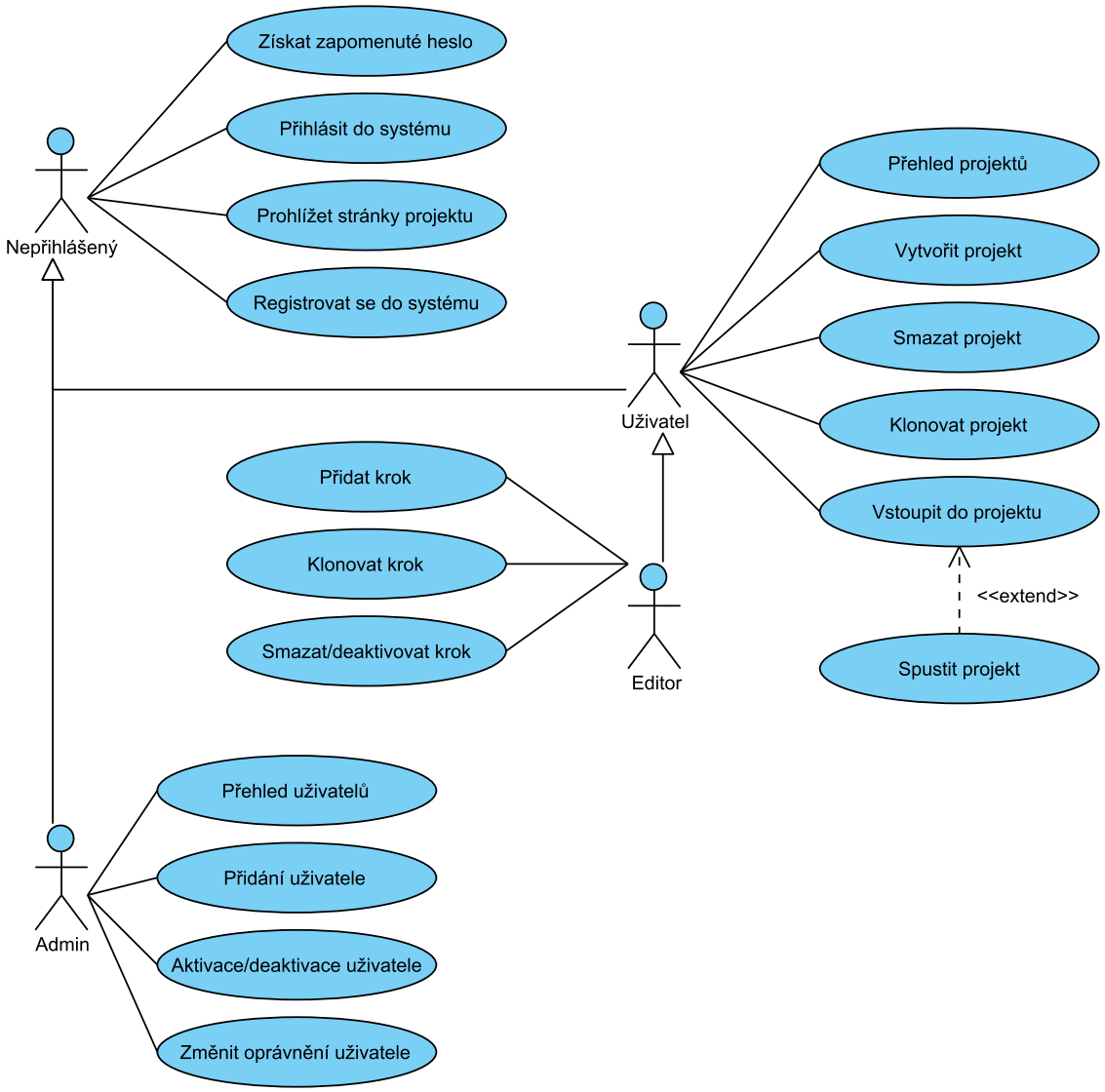
5.1 Diagram případů použití

Diagram užití (anglicky use case diagram) se v softwarovém inženýrství používá jako diagram chování definovaný jazykem UML (Unified Modeling Language) a pomáhá definovat hranice a rozsah systému. V tomto případě pomáhá přehledně zachytit všechny uživatelské role systému a akce, které mohou tyto role provádět během vzájemného dialogu se systémem.

Tento systém můžeme rozdělit pomocí čtyř uživatelských rolí, kde jedna z nich tvoří výjimku a to je role „*nepřihlášený uživatel*“ — s touto rolí ovšem musíme ve webovém portálu uvažovat a zvážit jaké sekce lze této roli zpřístupnit. Dalšími rolemi jsou pak role „*uživatel*“, „*editor*“ a „*administrátor*“, které jsou vyznačeny v diagramu na obrázku 5.1. Nyní si tyto role a akce, které mohou provádět, probereme detailněji.

Nepřihlášeného uživatele můžeme považovat za speciální roli z důvodu, že nemá žádný přístup k hlavní funkčnosti systému a jeho jedinou funkcí je možnost zobrazit veřejný obsah a sdělení portálu a umožňuje všem ostatním rolím se přes tuto roli do systému přihlásit. Pro tuto roli také musí být zavedena možnost se do systému registrovat pomocí svého e–mailu. Ten pak slouží jako ověření jedinečnosti uživatele a toho, že se do systému nesnaží registrovat automatizovaný program stejně tak pro možnost zaslání zapomenutého hesla pro nutnost přihlášení do systému — tato funkce je dnes již samozřejmou součástí každého webového portálu.

Každá ze tří dalších a hlavních rolí v systému přechází do své role po přihlášení z předchozí role *nepřihlášeného uživatele*, čímž nabývá svých specifických vlastností. První a zcela oddělenou rolí od ostatních dvou je role *administrátora*. Tato role plní funkci správce systému, který má možnost editovat seznam uživatelů portálu. Mezi její pravomoci patří



Obrázek 5.1: Diagram případů užití.

možnost změnit roli již registrovanému uživateli, nebo pozastavit možnost jeho přístupu k systému úplně tím, že změní status jeho aktivity na neaktivní. Rozhraní pro administraci systému plní pouze základní funkci a z hlediska systému další akce nejsou ze strany administrátora potřebné a mohou být tedy obsaženy v případném rozšíření systému.

Druhou a základní rolí je role *uživatele*, který po přihlášení získá standardní oprávnění přístupu k systému. U této role předpokládáme, že má v systému pouze přístup k již zavedeným krokům a z nich smí skládat vlastní projekty, z čehož plyne, že nejdůležitější akcí, kterou smí provést je tvorba a následná editace a mazání projektu. Tyto kroky jsou přímo spojeny s výpisem všech projektů, odkud se uživatel dostává ke všem dalším akcím. Akce pro spuštění projektu je dostupná pouze po vstupu do systému. Uživatel je osobou, která bude k systému přistupovat pouze za účelem pohodlné tvorby projektu a proto je tedy vhodné, aby byly akce zavedeny co nejúčelněji a byly implementovány pouze ty opravdu potřebné a nevznikal zmatek ve věcech, se kterými uživatel smí pracovat.

Rolí, která rozšiřuje roli předešlou, je role *editora*, který má na starosti oproti uživatelské roli navíc ještě vkládání kroků do systému. Stejně jako u projektů je ke krokům zavedena standardní množina akcí, kterými smí editor krok vytvořit, upravit či smazat nebo případně pouze editovat, pokud již nemá možnost krok smazat. U projektů (resp. kroků) je navíc přidána možnost jejich klonování, čímž se vytvoří nový projekt (resp. krok) přesně podle definice klonovaného projektu (resp. kroku).

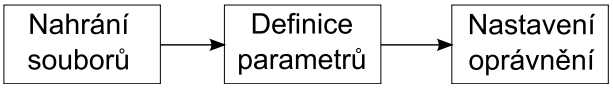
Pohledem na složitost diagramu použití zjistíme, že rozsah části prezentované webovou aplikací není příliš velký a lze tyto kategorie lehce dělit do několika málo kategorií a ty pak přehledně nabídnout v hlavním menu, čímž se výrazně zvýší uživatelův přehled o možnostech celého systému. Stejně tak v každé kategorii existuje pouze malý počet dalších akcí, které lze standardizovat a tím nabídnout pevně definované ovládací prvky, na které si může i méně zkušený uživatel rychle zvyknout. Jen dvě z akcí (tvorba/editace projektu a tvorba/editace kroku) jsou složeny z relativně složitějších celků, které ovšem lze samostatně strukturalizovat — tomuto tématu se budeme věnovat v následující sekci.

5.2 Rozvržení tvorby bloku a projektu

Z rozsahu aplikace, který nastínil diagram případů užití, můžeme vyvodit, že nejdůležitějšími prvky celého portálu je samotná tvorba a editace bloku či projektu. Vzhledem k tomu, že z pohledu systému je tvorba i editace projektu či bloku věcí naprosto stejnou a rozdíl je pouze v tom, zda se data do systému zavádí či se mění již stávající, budeme v rámci této sekce tento pojem uvádět pouze jako jediný a budeme mluvit pouze o *tvorbě* bloku či projektu.

5.2.1 Schéma tvorby bloku

Nejprve se tedy podívejme na schéma tvorby bloku (obrázek 5.2), který je potřebný pro možnost vytvoření projektu (připomeňme, že v rámci projektu již tento blok nazýváme krokem — v této sekci by mohlo dojít k záměně významů, a proto je vhodnější vyměnit tyto synonymní termíny).



Obrázek 5.2: Schéma postupu při tvorbě bloku.

Toto schéma je velice jednoduché a může být považováno i za schéma obrazovek, mezi kterými je třeba přejít během tvorby bloku. Uživatel je nucen v každém ze tří kroků zadat pouze nejnutnější věci a ostatní možnosti jsou defaultně nastaveny tak, aby v případě potřeby tvorba zabrala co nejkratší čas spolu s co nejmenším počtem kliknutí.

Nahrávání souborů

Prvním krokem při definici bloku je nahrávání spustitelných souborů, které jsou opěrným bodem celého bloku. Těchto programů může být více a je vhodné, aby měl uživatel možnost tyto programy pohodlně nahrát všechny společně. Populárním a moderním prvkem v oblasti webových aplikací se stal box, umožňující vložení souborů do stránky pouhým přetažením z jiného okna či jiné aplikace. Tento prvek je ve většině případů realizován čistě v programovacím jazyce JavaScript a odesílání na server není řešeno pomocí klasických HTML prvků ve formuláři pro nahrávání souborů. Ovšem toto řešení je značně komplikované při použití frameworku Nette pro jazyk PHP a obnášelo by vytvoření zcela nové komponenty pro tento framework, která se o danou věc samostatně stará. Toto řešení by ovšem přesahovalo rámec této práce a přitom stejného výsledku jde dosáhnout za pomoci klasických komponent, které nabízí k použití framework Nette a jen malého přispění JavaScriptového frameworku jQuery. Nastíněné řešení spočívá v možnosti změny vlastností klasického HTML prvku `<input type="file">` pro nahrávání souborů. Tento prvek umožňuje vložení souboru do prvku za pomoci přetažení myší z jeho aktuálního umístění. Stejně tak lze prvku nastavit atribut, který povolí možnost nahrávání více souborů. Poté již stačí prvku vhodně změnit vlastnosti pro zobrazení kombinací právě frameworku jQuery a kaskádových stylů v jazyce CSS. Tímto dosáhneme stejného výsledku jako při definici zcela jiného přístupu při použití jen JavaScriptového odesílání souborů na server a zároveň zůstává možnost nahrávání souborů stejně dostupná i bez zapnutého vykonávání JavaScriptu na straně webového klienta — změní se pouze vizuální vzhled tohoto prvku.

Po vložení a nahrání alespoň jednoho souboru aplikace zkontroluje, zda je soubor spustitelný a pokud nikoli, pak zobrazí pro tento nahraný soubor výstražnou ikonu, aby mohl uživatel rozpoznat možnou chybu a nahraný soubor zrušit. Není ovšem vhodné nahrávání souborů omezovat, neboť spustitelné soubory mohou být uloženy v neznámém formátu a pro ně by mohlo být nahrávání zbytečně omezováno. Po úspěšném vložení potřebných souborů může uživatel přejít do dalšího kroku a data jsou momentálně uložena do dočasné složky pro nahrávání souborů, společně s daty uloženými v `sessions` oblasti na straně PHP. Tímto je dosaženo toho, že se data zbytečně neukládají do databáze, dokud není tvorba bloku ukončena a zároveň se může uživatel kdykoli vrátit k jakémukoli ze tří kroků stejně tak, jako může odejít i do jiné sekce webové aplikace a v případě, že zůstane přihlášen, pak je nedokončený blok evidován v paměti a je vyvolán dotaz pro pokračování v něm při každém pokusu o tvorbu nového bloku.

Definice parametrů

Druhým krokem, který sebou nese mnohem více údajů k vyplnění, je pak definice parametrů pro vložené programy. Programů může být více z důvodu jejich kooperace a mohou generovat soubory, na které čeká program, který má být spuštěný následně za ním. Z tohoto důvodů musí být v tomto kroku řešeno pořadí programů, ve kterém se spustí v případě zadaného požadavku v rámci projektu. Změna pořadí vypsaných elementů ve webové stránce jde vyřešit za pomoci frameworku jQuery, který podporuje funkce přetažení myší, tzv. drag & drop metody. Za pomoci tohoto prvku jsme schopni uživateli nabídnout pohodlnou

možnost organizace programů bez nutnosti zadávat pořadí číselně a zpětně tyto hodnoty kontrolovat při odeslání. Stejně tak se může vyskytnout program, který může běžet paralelně k ostatním programům spouštěným v sérii a pokud uživatel již předem ví, že nemá žádné vazby s ostatními programy, je možné tento program spustit paralelně, což vyřešíme zaškrtávacím polem, které tento fakt bude v systému reflektovat.

Nyní se blíže podívejme na možnosti zadávání parametrů. Zadávaný program uvažujeme takový, který je vždy spouštěn s množinou parametrů, které mu předají všechny potřebné informace, podle kterých si je schopen vše další dohledat. Těmito parametry mohou být pouze textové hodnoty, které v systému dělíme do dvou skupin: názvy přepínačů a hodnoty přepínačů, kde hodnoty přepínačů mohou být jak textové hodnoty, tak cesta k souboru. Toto dělení není podstatné pro spouštěný program, neboť z jeho strany to vždy musí být pouze textové řetězce, ovšem z hlediska systému je situace již zcela jiná. Systém musí rozpoznat, jak má vloženou hodnotu kontrolovat a navíc je podstatné to, že pomocí parametru ukazujícího na umístění souboru jsme schopni dva oddělené bloky provázat.

Uveďme si stručně situaci, která bude popsána v následujících sekcích (počínaje sekcí 5.3 a dále) podrobně. Nahrané programy tvořící blok jsou uloženy v systému samostatně, aby jejich vykonávání neovlivnilo chod jiných bloků, a celý sled bloků je tedy identifikován vnitřním schématem uložení, které je v systému uloženo za pomoci identifikátorů. Z těchto identifikátorů je následně možné sestavit cestu od jakéhokoli bloku k jinému a tím poskytnout možnost programu přistoupit k žádanému programu. Vzhledem k tomu, že program může zpracovávat hned několik souborů z více bloků najednou, by bylo nevhodné přesměrovat program do jedné složky, ze které si bude program sám vybírat vstupní programy bez udání cesty parametrem.

Vzhledem k této situaci je tedy uživateli nabídnuta možnost zadat hodnotu přepínače společně s typem a přednastavenou volbou hodnoty. Pokud je hodnotou přepínače textový řetězec, pak je situace jasná, ale v případě, kdy má za hodnotu posloužit vstupní soubor, je potřeba přednastavit tento soubor ze strany editora, který blok vkládá do systému. Tomu je pro tyto účely umožněno soubor nahrát a tento soubor je pak v systému přednastaven jako výchozí volba pro vygenerování cesty k souboru, která se následně předá jako parametr. Obecně řečeno, pokud by v takto definované situaci editor vytvořil blok programů s pevně nastavenými parametry, poté by uživatel na straně tvorby projektů pouze mohl bloky vkládat, ale neměl by jedinou možnost ovlivnit běh těchto bloků, což může být v některém případě žádoucí, ale nevede to k žádané funkcionalitě systému.

U každého parametru je tedy evidována možnost ke změně onoho parametru ze strany projektu. Takovouto možnost nazýváme „editovatelný parametr“ a znamená, že pokud uživatel zařadí tento blok do svého projektu a bude daný blok editovat, pak bude mít zobrazeny pouze tyto parametry, u kterých následně bude moci hodnotu ovlivnit (toto téma však přenechme následující podsekci 5.2.2, která se mu věnuje blíže).

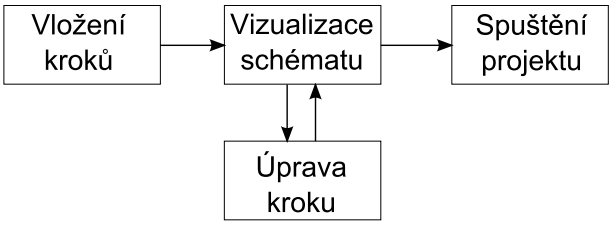
Poslední dodanou funkcionalitou na straně parametrů je pak možnost tzv. „nulového parametru“, kdy program očekává na zadané pozici parametru pouze hodnotu bez názvu přepínače. Pro tyto účely byla do systému navržena možnost, kdy editor vloží místo názvu parametru hodnotu `$null`, která způsobí, že název přepínače bude zcela vynechán v sestavované řádce pro spuštění a bude uvedena pouze hodnota tohoto přepínače. Můžeme navíc zmínit, že parametry mají své pevné pořadí, a proto je vhodné toto pevné pořadí uvažovat i ve vstupním formuláři, který navíc může pro efektivitu výpisu zobrazovat kolonku pro vyplnění dalšího parametru až v případě, kdy uživatel zadá hodnotu do předchozího parametru.

Nastavení oprávnění

Posledním krokem, který musí uživatel při definici bloku projít je krok pro nastavení oprávnění k přístupu k bloku. Tento krok pouze zobrazuje přepínače pro zavedení dvou možností. První z možností je, zda má být blok veřejný pro všechny uživatele v systému, nebo jej lze zobrazit pouze majiteli bloku. Druhou možností je to, zda má být krok po dokončení hned uvažován jako dokončený. Až příznak dokončení zařadí blok do seznamu použitelných bloků pro vložení do projektu, nehledě na to, zda je krok veřejný či nikoli. Tento příznak ovšem již zakáže úpravy bloku, stejně tak jako použití bloku v projektu zamezí možnosti tento příznak znovu odebrat, či blok zcela smazat. Smazání použitého bloku pouze blok odstraní ze seznamu použitelných bloků a již vytvořené projekty s tímto blokem tedy nebudou ohroženy na plné funkčnosti.

5.2.2 Schéma tvorby projektu

Ze schématu na obrázku 5.3 můžeme vidět, jak jsou rozděleny a navrženy kroky při tvorbě projektu. Práce na projektu byla navrhována tak, aby uživatel mohl nezávisle přecházet z jednoho kroku do druhého či se případně mohl vracet zpět a všechna data byla ukládána do systému až po potvrzení daného kroku bez ztráty dat, pokud tvorbu opustí v jakékoli fázi. Tvorba projektu je rozvržena tak, aby uživatel měnil data po malých částech a ty musel pro uložení odeslat na server, čímž se důraz přenesl na ukládání dat po částech bez nutnosti kontroly, zda je celý projekt již hotový a odeslaný ke zpracování. Tento přístup má výhodu v tom, že není třeba data ukládat do paměti a přitom není sníženo uživatelské pohodlí. To naopak získá tím, že má uživatel možnost provádět zkušební změny na malém úseku a pokud s nimi není spokojen, může je kdykoli zrušit tím, že změny nepotvrdí a vrátí se zpět.



Obrázek 5.3: Schéma postupu při tvorbě projektu.

Vložení kroků

Základním úkolem, bez kterého se tvorba projektu neobejde, je vložení bloků (připomeňme, že v rámci projektu těmto blokům říkáme kroky projektu), které tvoří stavební kameny každého projektu. Uživatel má k dispozici dva seznamy již přednastavených bloků, kde prvním je seznam jeho vlastních bloků (které smí vytvořit pouze s dostatečným oprávněním) a druhým seznamem jsou bloky definované a zveřejněné jinými uživateli. Z těchto bloků nyní potřebuje vybrat ty, které zařadí do svého projektu, přičemž má možnost projekt vytvořit z více totožných bloků. Tento problém je v aplikaci řešen tím, že při výběru bloku a jeho zařazení do projektu se vytvoří jeho kopie s odkazem na originální umístění bloku, čímž vznikne i vazba na blok, která zablokuje možnost smazání tohoto bloku a povolí pouze jeho smazání v seznamu bloků, ze kterého byl vybrán.

Po vložení všech potřebných bloků může uživatel přejít k dalšímu kroku, kterým je vizualizace schématu. Při vrácení se na tento krok může uživatel kdykoli přidat do schématu další blok, případně odebrat libovolný z již zařazených, přičemž se zruší všechny změny, týkající se daného bloku.

Vizualizace schématu

Další postup tvorby projektu byl vytvořen za pomoci vizualizace schématu projektu, které lze v průběhu editace upravovat. Toto schéma je zpřístupněno jako základní obrazovka tvorby projektu a lze z něho přejít do všech fází tvorby projektu, stejně tak jako se do něj z jiných fází lze vrátit. Díky vizualizaci v podobě souborového formátu `svg`, vloženého přímo do stránky, lze zpřístupnit jednotlivé bloky jako odkazy, díky nimž se uživatel přepne do editační části vybraného kroku. Toto schéma prezentuje uspořádání kroků a jejich programů, které tvoří jednotlivé řádky daného kroku. Pokud některý z programů obsahuje parametr, čekající na soubor vygenerovaný jiným krokem, pak je vedena šipka od programu směrem k bloku, který hledaný soubor, či složku generuje. Tato šipka reprezentuje orientovanou hranu a kroky projektu reprezentují ve vytvořeném grafu jednotlivé uzly. Na každou hranou je pak uvedena strukturovaná řádka obsahující informaci o názvu parametru, ze kterého hrana vychází, a také jeho typ.

Schéma je generováno programem `dot` ze série programů *Graphviz* a vytvořené schéma je předáno stránce jako výstup tohoto programu. Generování tohoto výstupu probíhá v rámci milisekund a je tedy srovnatelné s dobou potřebnou pro generování samotné stránky v PHP. Potřebná datová struktura pro vygenerování tohoto výstupu je předem zkomponována v rámci generování stránky a pouze tato data vedoucí k vytvoření aktuálního grafu jsou uloženy jako samostatný soubor.

Kroky ve schématu jsou řazeny zleva doprava, což znamená, že blok, který nemusí čekat na spuštění žádného kroku před ním, bude umístěn vždy více vlevo než kroky, které jsou na něm závislé, z čehož plyne jasná posloupnost spouštění programů. Díky rychlému generování schématu však můžeme uživateli nabídnout možnost generování schématu jiným směrem, pokud by mu stávající směr zarovnání nevyhovoval, nebo by jiná orientace poskytla lepší přehled, který je pro sestavení projektu v kratším čase důležitý. Proto byly navrhnuty prvky, které tuto změnu umožní a přímo aktualizují schéma podle nového směru zarovnání. Nabídnuta je rotace směru doprava po směru, stejně tak jako doleva proti směru hodinových ručiček společně s možností nastavení standardního stavu, neboť směr rotace se ukládá pro přihlášeného uživatele po dobu jeho práce na stanici, aby v nastavené výhodné rotaci mohl přímo generovat i ostatní schémata z jiných projektů.

Úprava kroku

Krokem tvorby projektu, který je stejně četně zobrazován, jako generování schématu je i úprava vybraného bloku neboli kroku projektu. Pokud uživatel klikne na jeden z kroků projektu, je mu umožněna úprava tohoto kroku, kde jsou mu nabídnuty k editaci pouze programy, které souvisí s parametry, u nichž byla vybrána možnost editace.

S četností zobrazování tohoto kroku souvisí také nutnost dobré členitosti a přehlednosti všech zobrazených prvků na stránce. Jednotlivé parametry tedy byly v aplikaci viditelně a přehledně odděleny a ke každému z prvků byla přidělena stejná množina ovládacích prvků společně s prvky pro vkládání dat. Z předchozích sekcí víme o parametrech programů to, že mohou být dvojího typu (pro hodnotu a pro soubor) a dále se jeden z těchto typů může sestávat z více podružných informací (u parametru pro soubor je třeba zadat název

hledané složky či souboru). Pokud bychom všechny tyto možnosti vypsali do statického formuláře, který se celý zobrazí po načtení stránky, vznikl by v zadávaných možnostech zmatek a formulářů pro vyplnění by bylo příliš mnoho s tím, že by spoustu z nich bylo zobrazeno zbytečně a uživatel by sám musel hlídat, které vyplnil a které by vyplnit měl. Další možností pak je rozdělit tuto složitou obrazovku na několik dílčích obrazovek, mezi kterými se uživatel postupně přepíná po zadání vhodných dat, a po jejich kontrole jsou zobrazeny patřičné další prvky. Ovšem tato situace zdržuje čas, který musí uživatel strávit čekáním na odeslání formuláře a přijetí nové stránky. Pro tyto problémy nabízí efektivní řešení právě použitý framework Nette, který vyniká v práci s formuláři — implementuje totiž v základu podporu JavaScriptového frameworku jQuery, čímž umožňuje skrývat či zviditelňovat jednotlivé položky formuláře na základě kontroly obsahu jiného formulářového prvku nehledě na pořadí prvků ve formuláři. Stejná výhoda frameworku byla využita již ve fázi tvorby bloku, kdy se stejným principem dalo provést zobrazování nového řádku pro parametr pouze při vyplnění dat do předchozího řádku. V této fázi stejným principem dosáhneme výrazného přispění k orientaci ve složitém formuláři a v interakci s uživatelovým výběrem zobrazujeme pouze možnosti, které může vyplnit. Po odeslání formuláře je již snadnou věcí data vybrat pouze podle odeslaných voleb a jiná zadaná data odfiltrovat.

Pro jednotný styl celé aplikace byl pak vybrán stejný vizuální styl formuláře, který byl implementován i v obdobném formuláři, který sloužil pro samotnou definici bloku.

Spuštění projektu

V případě, že uživatel provedl všechny potřebné úkony v editaci schématu, může přejít ke spuštění projektu. Vzhledem k tomu, že počet kroků v projektu, stejně jako počet programů v jediném bloku není v systému nijak shora omezen, může být tato část obecně výpočetně velmi zdlouhavá a nikdy nemůžeme předem odhadnout, jak dlouho bude výpočet trvat, či jaké chyby se během jeho spuštění mohou vyskytnout. Z tohoto důvodu je nepraktické spouštět program v době generování samotné stránky projektu. Proto je potřeba předat vyřízení požadavků pro zpracování jiné aplikaci, která se o zpracování zaslaných dat postará a nebude žádným způsobem omezovat generování stránek webového portálu. Celým tímto problémem se podrobně zaobírají bezprostředně následující sekce této kapitoly, a proto se nyní zmiňme o možnostech výpisu dat po předpokládaném zpracování odeslaných dat.

Pro tento příklad stručně uveďme, že všechna potřebná vygenerovaná data jsou uložena společně s chybovými hlášeními, či standardním výstupem programů ve struktuře složek, do kterých má webový portál přímý přístup. Pro informaci o průběhu musí být tedy uživatel spraven o průběhu všech programů, a proto jsou mu po úspěšném provedení programu zobrazeny přehledně a členitě data, která jsou vytištěna na standardní či chybový výstup, přičemž typ zprávy je příslušně vizuálně označen společně s jasným oddělením, ke kterému programu výpis patří. Vrácení vygenerovaných souborů probíhá pomocí zabalení složek do archivu, který je uživateli taktéž zpřístupněn ke stažení ve standardním archivačním formátu `zip`. Během generování veškerých výstupů projektu je možnost spuštění projektu blokována a je zobrazena pouze stránka oznamující, že je projekt vykonáván a v případě, že uživatel na stránce čeká, jsou odesílány dotazy technologií AJAX na server pro průběžnou aktualizaci stránky a v případě dokončení projektu jsou na stránku automaticky načteny aktuální data.

Pokud uživatel projekt opustí, data budou uchována v rámci tohoto projektu až do dalšího spuštění projektu, který stávající data přepíše novými podle aktuálního, případně i změněného schématu projektu. V případě, že uživatel ve výpisu projektů zvolí možnost

klonování projektu, je tento projekt klonován bez vygenerovaných souborů a nový projekt tak musí být opětovně spuštěn, což je navrženo kvůli časové úspoře klonování projektů společně s tím, že klonování je navrženo především pro vytváření projektů z jednoduchých šablon, které slouží především jako opora pro tvorbu dalšího projektu a je tedy zbytečné kopírovat již vygenerované soubory.

5.3 Rozdělení aplikace

Webový portál jako takový potřebuje na serverové straně aplikaci, která se stará o vytváření webových stránek — v případě této práce je to pak PHP, které sestavuje stránky podle zasílaných dotazů. Serverová strana má tedy jen omezený čas na vrácení odpovědi, během kterého musí provést všechny úkony, o které je žádána. Náš problém spočívá v potřebě spustit obecně velké množství programů a počkat, až budou všechny dokončeny. V případě malého počtu programů, které mají velmi krátký běh je možné tyto programy volat přímo ze strany PHP, ovšem v našem případě to již možné není, protože není nijak zaručena doba běhu, a proto tedy musíme zvolit jiné řešení jak programy spouštět.

V návrhu aplikace byla zvolena možnost zapsání instrukcí do souboru společně s uložením potřebných dat do vhodné adresářové struktury a vytvořit zcela samostatný program, který si sám zjistí, zda existují potřebné instrukce pro spuštění a podle toho tyto instrukce vykoná a vrátí výsledky do předem určené adresářové struktury, kde bude webová aplikace očekávat data ke zobrazení výsledku. Z tohoto přístupu vyplývá, že celý systém musíme rozdělit na dvě části a jim zajistit pouze vhodné prostředí, kam budou ukládat data a zprávy, pomocí nichž budou vzájemně komunikovat. Částí, která bude uživatelům webového portálu přístupná, je část *PHP aplikace*, která bude generovat obsah webu a zároveň podle uživatelských požadavků generovat inicializační soubory, ze kterých bude čerpat druhá část, kterou je *program v jazyce C++*, jenž bude obstarávat samotné spouštění požadovaných souborů v projektu a starat se o to, aby vrácená data byla přístupná zpětně z první části celého systému.

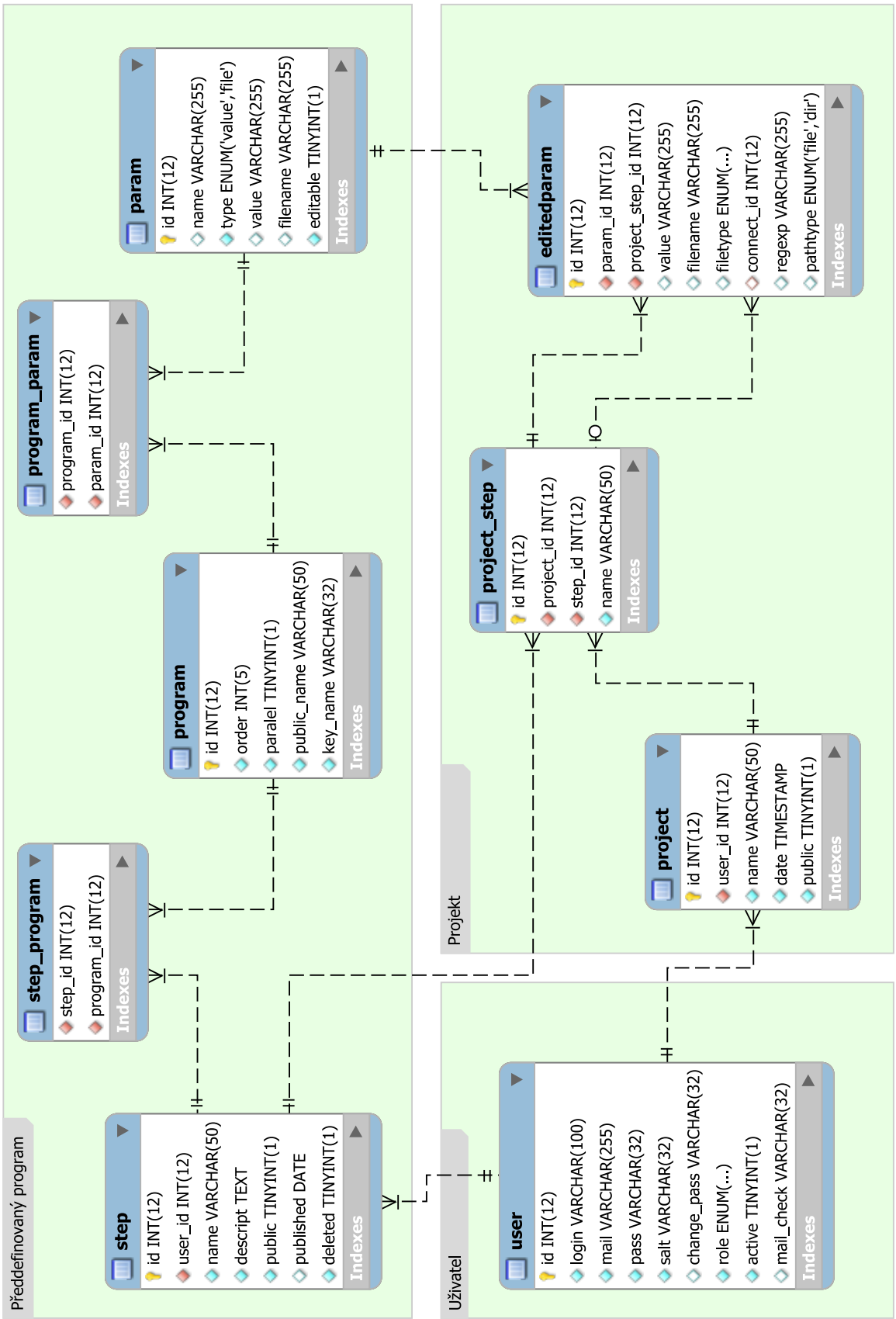
Vzhledem k faktu, že veškerá data a jejich interpretace vzniká na straně webového klienta, začneme tedy s popisem částí stylem, kdy nejdříve vysvětlíme způsob vnitřní interpretace dat na straně webového klienta, společně se všemi úkoly, které obnáší transformace interních dat na instrukce pro druhou část systému. Po sekci popisující tuto první část tedy přejdeme k pohledu zpracování těchto instrukcí programem v C++, který tvoří právě druhou část systému, a navíc v této sekci objasníme, jakým způsobem tento program svoji práci provádí.

5.4 PHP aplikace

Tato sekce se věnuje popisu interní reprezentace dat v systému na straně PHP aplikace a jejím úkolem je představit podrobně všechny důležité části, kterými je především styl uchovávání dat v systému pro bloky a projekty a dále tvorba schématu projektu, ze kterého musíme být schopni následně získat potřebné informace pro správné zpracování projektu.

5.4.1 Uchovávání interních dat

Webový portál z hlediska uložených dat lze rozdělit na tři celky, kde prvním je část, kterou tvoří uživatelé systému, druhou částí bloky a poslední částí pak projekty. Obdobně je strukturován i Entity-relationship (dále jen zkráceně ER diagram) diagram sloužící k abstrakt-



Obrázek 5.4: Entity-relationship diagram.

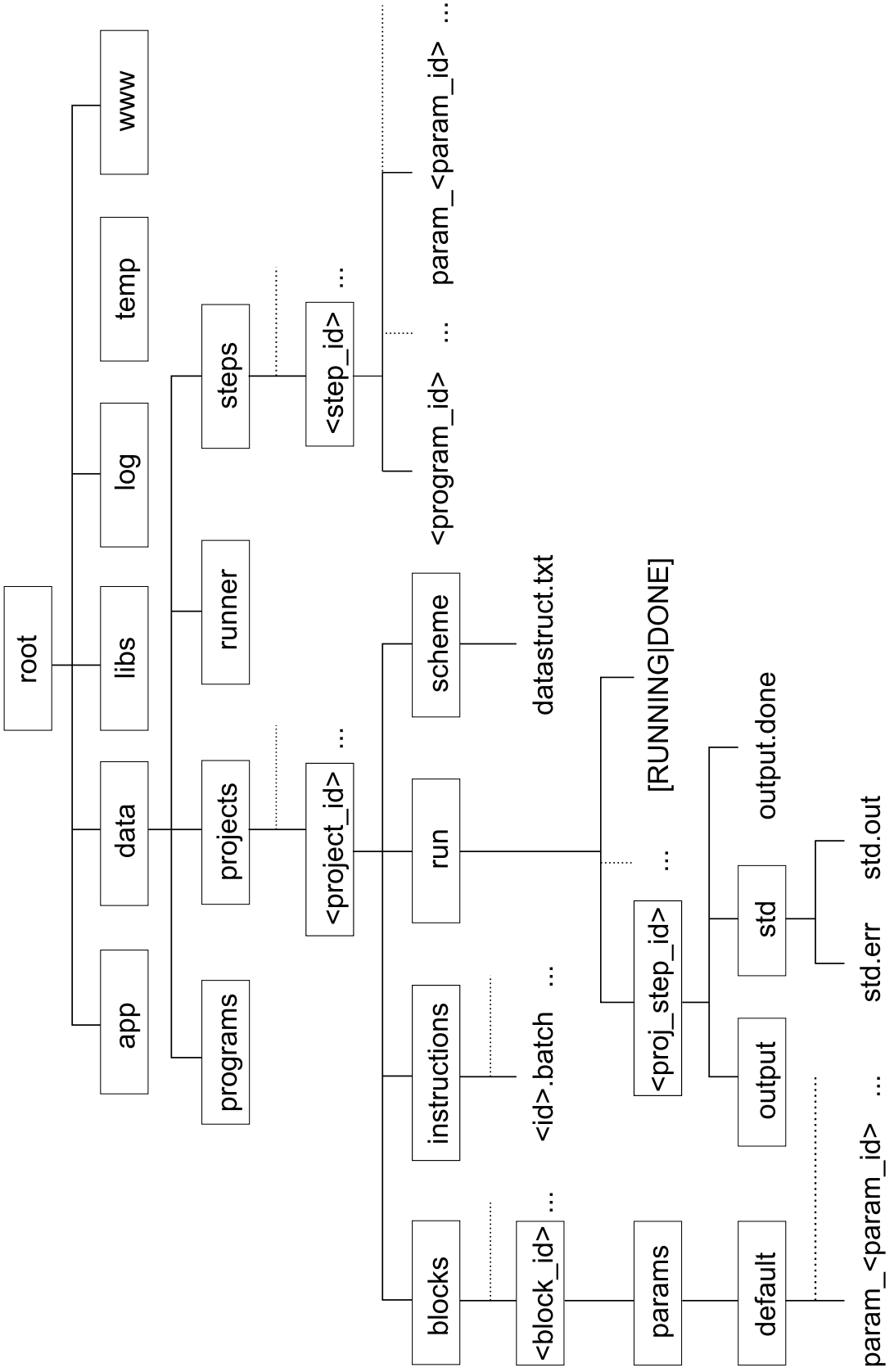
nímu a konceptuálnímu vyobrazení dat na obrázku 5.4. Tato datová struktura je interně reprezentována MySQL databází, ze které jsou potřebná data získávána. Webová aplikace do této struktury ovšem ukládá veškeré podstatné informace k evidenci uživatelů, bloků i projektů a vztahů mezi nimi, ovšem pro bloky je situace komplikovanější v tom, že společně se záznamem v databázi také musí být evidován spustitelný program, který ovšem nemůže být uložen jako záznam v databázi. Stejně tak budou tyto programy generovat další a to předem nedefinované a nestrukturované soubory či složky a v tomto případě takováto data nelze do databáze uložit. Z tohoto důvodu je třeba programy ukládat samostatně v souborovém systému a v databázi vést pouze evidenci o umístění těchto souborů a jejich vazbách k jiným prvkům, případně evidovat spojení, která se mezi těmito programy vytvoří během tvorby projektu. Také vzhledem k nemožnosti ukládání výstupních souborů spouštěných programů musíme předem specifikovat a rozvrhnout adresářovou strukturu tak, aby byla všechna data dohledatelná z webové aplikace a nedocházelo ke ztrátám vygenerovaných dat či k přepisu důležitých souborů.

Tedy pokud následující situaci vztáhneme k uvedenému ER diagramu, pak zjistíme, že tabulky `program` a `param` musí být doplněny o případný program, či soubor. Tabulkou `program` jsou reprezentovány dílčí programy, jež jsou součástí každého bloku a atributem `key_name` je pak tento záznam provázán se souborem, který je uložen na příslušném místě v adresářové struktuře. Vzhledem k tomu, že názvy nahrávaných programů mohou být obecně jakékoli a to i shodné, pak je třeba přejmenovávat tyto programy za pomoci vytvořeného jedinečného názvu pomocí vnitřní logiky aplikace v PHP. Obdobná situace nastává i pro tabulku `param`, ve které jsou uloženy záznamy o parametrech příslušících k danému programu. Parametry, u nichž je zvolen jako typ soubor, musí být navázány na soubor, který uživatel při definici tohoto parametru zadal jako výchozí možnost pro spuštění programu beze změny v projektu. Taktéž i tento soubor musí být uložen v souborovém systému pod unikátním jménem pro daný blok a toto jméno je v databázi uloženo pod atributem `filename`.

K celému pochopení ER diagramu uveďme ještě informaci o tom, že projekt je v databázi ukládán jako jediný záznam v tabulce `project` společně odkazem do tabulky `project_step`, která uchovává informace o krocích, které byly do projektu zařazeny, ovšem se záznamem v této tabulce již nesouvisí nutnost mít uložen taktéž soubor v příslušné složce, neboť program jsou volány z umístění, které je uvedeno v tabulce `step`. Což ve výsledku znamená to, že spouštěný program v projektu je volán stále ze stejného umístění, stejně jako jeho výchozí parametry, čímž dochází k úspoře místa na serverovém disku a parametry, které jsou změněny u vybraného kroku v projektu, se pouze promítnou v tabulce `editedparam`, kde dochází k ukládání všech provedených změn u vybraného parametru — touto změnou je například reference od parametru k bloku, ve kterém má být hledán vstupní soubor a právě na tuto možnost odkazuje atribut `connect_id`, který plní funkci nepovinného cizího klíče.

5.4.2 Adresářová struktura pro podporu databáze

Z textu v předchozích odstavcích víme, že data uložená v databázi souvisí se soubory uloženými na disku serverové stanice. Na tyto data nevede přímá vazba z databáze a proto je tedy nutné, aby byla tato data správně organizována ve složce, ke které bude přistupovat pouze webová aplikace, neboť jen v ní je implementována logika ukládání dat do databáze i do adresářové struktury. Na obrázku 5.5 je znázorněno schéma adresářové struktury, která je nutná pro ukládání potřebných dat. Toto schéma ukazuje adresářovou strukturu za pomoci stromového grafu, který má intuitivní cestou vysvětlit čtenáři systém ukládání



Obrázek 5.5: Adresářová stuktura pro ukládání souborů.

na disk. Kořenem tohoto grafu je abstraktní složka root, ve které je celá aplikace na disku umístěna a za tuto složku již celý webový portál nemůže mít přístup. Každý uzel v grafu reprezentuje složku a vizuálně je reprezentován obdélníkem, listy jsou pak označeny pouze názvem bez obdélníku a reprezentují soubory, které jsou z hlediska aplikace stálé a pro správnou funkčnost systému důležité. Některé názvy složek či souborů obsahují špičaté závorky, které reprezentují proměnnou hodnotu a v případě, že těchto složek či souborů může být na stejné úrovni více, pak tuto situaci uvádíme třemi tečkami, vyskytujícími se bezprostředně vpravo od zobrazení složky či souboru. Hranaté závorky oproti tomu označují více vzájemně výlučných názvů pro existenci jediného souboru či složky, kde názvy jsou odděleny znakem `|`. Tedy v tomto konkrétním případě ve složce *run* bude několik složek pojmenovaných podle identifikátoru kroku z projektu (na obrázku 5.4 je tento identifikátor v tabulce `project_step` primárním klíčem `id`), který daná složka reprezentuje a pak zde bude jediný soubor a to buďto soubor se jménem `RUNNING` nebo `DONE`. Význam těchto souborů bude blíže specifikován v sekci 5.5 společně s dalšími důležitými složkami, které jsou v systému generovány a slouží ke komunikaci webového portálu s programem na spouštění projektů.

Složky první úrovně grafu jsou standardními složkami, které ke svému běhu vyžaduje a jak je definuje framework Nette (jediná složka `www` je zobrazitelná webovým klientem a obsahuje pouze soubor `index.php`) a jedinou výjimku tvoří přidaná složka `data`. V této složce musí být před spuštěním systému vytvořeny alespoň složky `projects`, `steps` a `runner`, ve které je již umístěn program pro spouštění projektů, jehož popis je obsahem sekce 5.5. Do složky `steps` se webovou aplikací vytváří složky obsahující spustitelné soubory, a do složky `projects` se obdobně ukládají veškerá data potřebná pro spuštění projektu, stejně tak jako data, která jsou spuštěním projektu vygenerována.

5.4.3 Tvorba grafu a vyhledávání

Další důležitou součástí PHP aplikace je tvorba a vyhledávání v datech, která tvoří graf. Při tvorbě a editaci projektu dochází pouze ke změně a ukládání faktů, které reprezentují pouze jednotlivé kroky projektu. Tedy ve skutečnosti nevytváříme nová spojení, ale pouze u jednoho kroku zadáváme takové vlastnosti, které toto spojení reprezentují, a je pak v režii aplikace, aby tato data oddělila a vykreslila.

Program `dot` vyžaduje pro správné generování grafu na svém vstupu instrukční soubor, který obsahuje informace v následující struktuře:

```
digraph "New Project" {
    graph [ rankdir = "LR" ];
    node [ shape = record ];
    edge [ ];
    node_90 [
        label = "<head>block 90 | <prog_60> program.exe"
        URL = "/project/block/90"
    ];
    node_91 [
        label = "<head>block 91 | <prog_60> program.exe"
        URL = "/project/block/91"
    ];
    node_90:head -> node_91:prog_60 [ label = "-param1" ];
}
```

Ze které můžeme vidět, že nejdůležitějším úkolem je předat v instrukčním souboru seznam uzlů a patřičných hran. Rozbor syntaxe tohoto souboru nepatří do náplně této práce a je dostupný na [6], navíc zde tato ukázka slouží pouze pro ilustraci, jak jsou data ve výsledku vypisována. Pro navrženou funkčnost je důležitý parametr *URL* v definici uzlu, neboť ten způsobí, že ve výstupním obraze ve formátu `svg` je uzel veden jako odkaz, přes který se přistupuje k editaci bloku. Další výpis dat do instrukčního souboru je již intuitivní a lze je lehce vyhledat v databázi.

Komplikovanější částí je ovšem samotné hledání ve schématu projektu. Zde máme dva úkoly k řešení: prvním z nich je nalezení smyček v grafu (popsáno v sekci 4.1.3) a druhým pak odlišení skupin kroků, které jsou na sobě závislé od skupin, které mohou být prováděny paralelně s nimi.

Hledání smyček

Oba problémy musejí vycházet ze stejných dat, která jsou přístupná z databáze a jsou případné definice propojení. Tato propojení jsou pouze eventualitou a jsou tedy vedeny pouze jako atributy u těch parametrů bloku, kde je dané spojení zavedeno místo výchozí hodnoty, kterou je nahraný soubor. Tedy prvním společným pod úkolem je vynětí všech vazeb na bloky pro každý blok daného schématu.

Tyto vazby si uložíme jako pole, které obsahuje tolik prvků, kolik je ve schématu bloků a v každém prvku dále obsahuje pole s identifikací bloku, na který je vázán (tedy na který blok čeká, aby se sám mohl spustit). V rámci PHP aplikace je implementována logika, která každý prvek prochází a rekurzivně do svého pole přidává i bloky, na které je vázán nějaký z již navázaných bloků — takto je pokračováno, dokud pole pro přidávání bloků již neobsahuje všechny bloky ze schématu či dva kroky za sebou není toto pole stejné. Tím tedy pro každý blok získáme pole obsahující všechny bloky, které musí být vykonány před spuštěním tohoto bloku. Pokud ovšem pole jakéhokoli bloku obsahuje i blok, ze kterého je vycházeno, pak je jisté, že blok čeká sám na své provedení, což je definice uváznutí a můžeme tedy schéma označit za neproveditelné díky výskytu smyčky.

Hledání skupin

Jak bude níže zmíněno, program pro spouštění projektů je schopen vykonávat části projektu paralelně. Ovšem již při definici souborů pro spouštěcí program již musíme vědět, které bloky jsou na sobě zcela nezávislé. Tento problém vychází částečně z předchozího problému, kdy přejímá pole obsahující pro každý blok pole bloků, které musejí být vykonány před ním a na tomto poli staví další svoji logiku.

Touto další logikou rozumíme vytvoření skupin bloků, které na sebe vzájemně čekají. Tedy pokud víme, že blok `3` čeká na bloky `1` a `2` a zároveň blok `5` čeká na bloky `3` a `4`, pak můžeme pro blok `3` i `5` vytvořit stejnou skupinu, která obsahuje všechny zmíněné prvky. Dvojnásobných projetím všech prvků výchozího pole tedy získáme výslednou skupinu pro každý prvek, kde našim žádaným výsledkem jsou pouze jedinečné skupiny. Po této operaci můžeme vytvořit pro každou tuto skupinu samostatný instrukční soubor a víme, že mezi bloky z tohoto souboru již není žádná vazba a mohou být tyto série provedeny paralelně.

Řazení bloků ve skupině

Posledním úkolem je seřazení bloků ve skupině, které slouží k tomu, aby se programy spouštěly v pořadí, kdy už jsou před nimi provedeny všechny programy, od kterých program

očekává soubory ke zpracování. Tento úkol může vyjít z dat, která generují dva předchozí úkoly, tedy z definovaných propojení mezi bloky a definovaných skupin.

Tento algoritmus tedy prochází již skupinu bloků, která tvoří podmnožinu všech bloků schématu. V této skupině je vždy alespoň jeden program, který nemá žádnou vazbu na jiný blok a tedy neočekává na svém vstupu soubor vygenerovaný jiným blokem (v případě, že by blok, bez propojení neexistoval, pak by se v této skupině vyskytovala smyčka). Tyto bloky jsou tedy pojaty za prvotní kroky a přímo zařazeny do výstupního pole jako první položka. Toto výstupní pole je následně rekurzivně doplňováno o bloky, které očekávají jako vstup pouze soubory od bloků, které jsou již ve výstupním poli zařazeny. Tímto způsobem se prohledá celá skupina dle vazeb a výstupní pole je předáno v okamžiku, kdy jsou do něj zařazeny všechny prvky skupiny, či v případě chybné identifikace skupiny jsou dva kroky za sebou stejné (tato podmínka je zavedena pouze pro jistotu ukončení `while` cyklu).

Díky výsledné posloupnosti lze identifikovat, které bloky lze vykonat i v rámci skupiny paralelně a tím tedy urychlit vykonávání projektu.

5.5 Spouštěcí programy v C++

V předchozím textu bylo zmíněno již vše, proč je potřeba programy spouštět mimo aplikaci psanou v jazyce PHP, nyní přejdeme tedy k důležitým faktům, které vedly k implementaci problému spouštění v jazyce C++ a co všechno požadovaná aplikace musí umět kromě spouštění zadaných programů.

Vzhledem k tomu, že v systému může být evidováno libovolné množství projektů, které dále mohou využívat libovolné množství programů, je třeba spouštění těchto programů rozvrhnout tak, aby čas pro spuštění byl co nejmenší a bylo možné využít všechny prostředky systému na zpracování. Na serverových stanicích je často využíváno více procesorů, a proto je vhodné, aby tento fakt byl brán i ze strany aplikace spouštějící programy projektu — tato aplikace byla nazvána v rámci této práce *runner*. Z předchozí sekce víme, že PHP aplikace je schopna vygenerovat po spuštění projektu soubory, které obsahují kroky projektu seřazené tak, jak je můžeme vykonat. Tyto kroky už je triviální přetransformovat do podoby příkazové řádky, která bude vykonána, neboť vždy známe cestu k programu, který se má vykonávat a jsme schopni podle dat v databázi sestavit i předávané parametry, kde mezi ty komplikovanější patří odkaz na složku nebo soubor, který bude vygenerován v navázaném bloku.

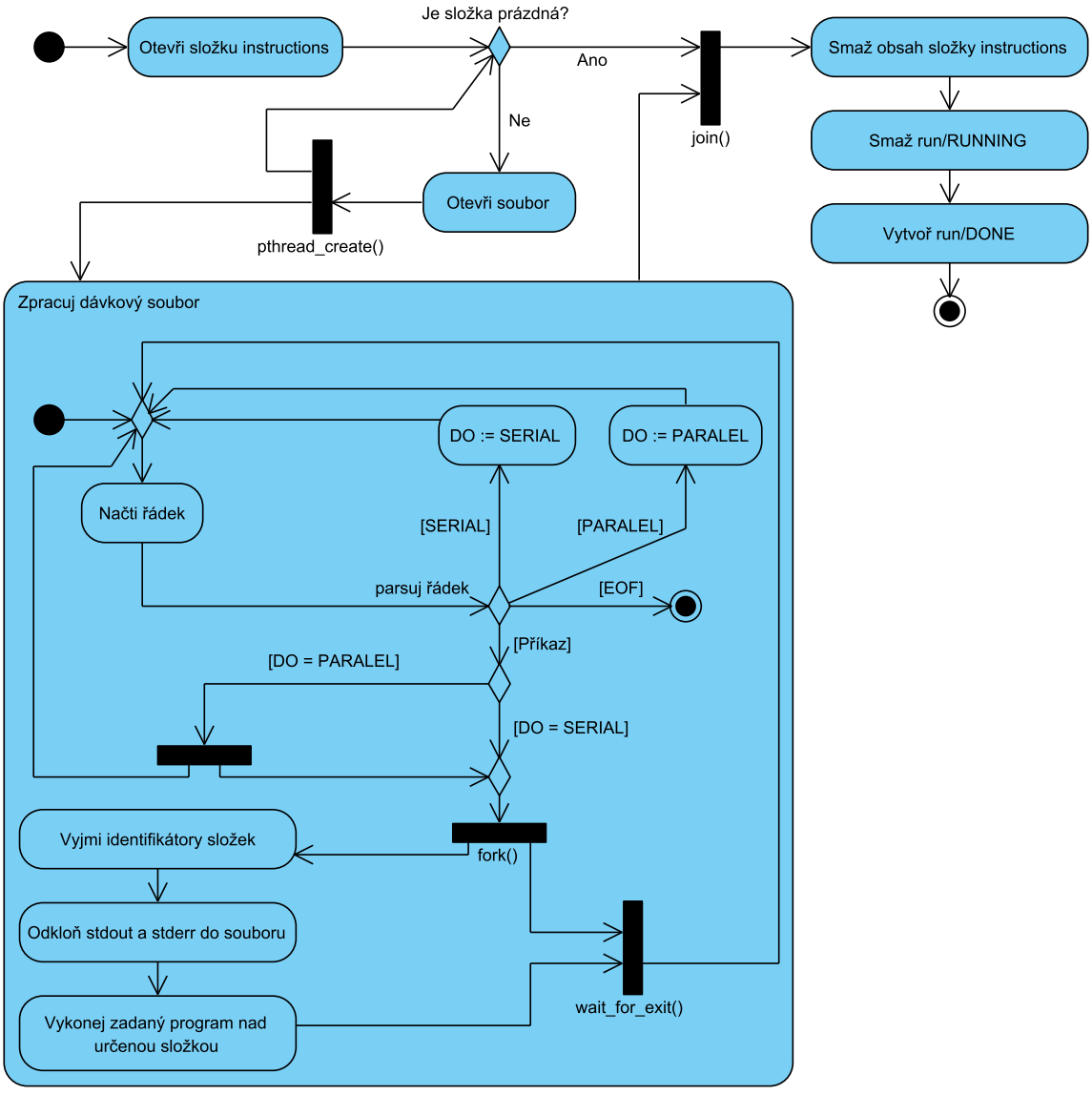
Bylo tedy navrženo schéma, kdy je PHP aplikací do složky `instructions` vloženo tolik dávkových souborů, kolik je ve spouštěném projektu paralelních skupin. Tyto soubory jsou pak identifikovány pořadovým číslem a vždy mají příponu `.batch`. Tento textový soubor vždy obsahuje data definovaná podle stejné struktury, kde jsou data čtena po řádcích a každý z řádků buďto obsahuje jedno z klíčových slov `SERIAL` nebo `PARALEL`, nebo řádek specifikující příkaz pro spuštění zadaný ve formátu:

```
<project_id>|<proj_step_id>|../cesta/spustitelny_soubor [parametr, ...]
```

Kde project_id i proj_step_id se rozumí identifikátor příslušné složky (viz schéma složek 5.5). Klíčová slova `SERIAL` a `PARALEL` pak složí k určení, zda se mají příkazy následující na řádcích za nimi zpracovávat sériově či paralelně. Ostatní řádky, které nesplňují tuto strukturu, mohou být při zpracovávání ignorovány a taktéž ze strany PHP aplikace nejsou generovány.

5.5.1 Runner

Program s pracovním názvem *Runner* je nejlépe vystižen pomocí diagramu aktivit 5.6. Pro spuštění tohoto programu je vyžadován jeden číselný parametr, kterým je identifikátor projektu pro spuštění. Podle tohoto parametru se program vykonává nad složkou, příslušející předanému identifikátoru projektu. V této složce je již připravena složka s názvem `instructions` a v ní dávkové programy s příponou `.batch`, které byly vygenerovány podle schématu projektu, jak je popsáno v odstavci výše.



Obrázek 5.6: Diagram aktivit pro program *Runner*.

Dokud tato složka není prázdná, jsou postupně tyto programy otvírány a paralelně zpracovávány. Po zpracování posledního souboru je obsah složky `instructions` vymazán a místo souboru `RUNNING` ve složce `run` je do této složky nahrán soubor `DONE`. Pomocí existence těchto souborů probíhá komunikace mezi aplikací v PHP a nyní popisovaným programem. Pokud strana PHP aplikace vygeneruje potřebné soubory pro běh *Runneru* je

vytvořen také soubor `RUNNING`, který indikuje zadání této skutečnosti a ze strany PHP již nemůže dojít k modifikaci této složky, dokud tento soubor existuje. Taktéž by bylo obtížnější kontrolovat, zda *Runner* již vykonal zadané úkoly, a proto po dokončení své práce vytvoří soubor `DONE` a soubor `RUNNING` vymaže. Obsah těchto souborů není pro funkci systému důležitý, ale pro případ dalšího rozšíření je do něho vkládána časová značka okamžiku vytvoření tohoto souboru. V případě souboru `DONE` slouží tato časová značka pro zobrazení času posledního dokončení projektu.

Zpracování samostatného dávkového souboru poté záleží na obsahu tohoto souboru. Čtení probíhá po řádcích, a pokud je indikováno na řádku klíčové slovo `SERIAL`, pak další řádky do výskytu klíčového slova `PARALEL` jsou zpracovávány sériově a po výskytu klíčového slova `PARALEL` je pro vykonání každého příkazu voláno nové vlákno. Samotné vykonání jednoho řádku dávkového souboru pak probíhá stejně, nehledě na to, zda vykonávání běží v samostatném vlákně, či vykonání dalšího řádku čeká na ukončení spuštěného programu identifikovaného na aktuálním řádku.

Každý řádek má uveden dva prefixy indikující identifikátor projektu a kroku tohoto projektu. Názvy těchto složek jsou důležité k tomu, aby *Runner* dovedl přesměrovat veškeré výstupy spouštěného programu do předem určených složek. K tomu je využito především funkce `dup()` jazyka C++, pomocí které jsme schopni přesměrovat standardní výstup programu do souboru `std.out` a chybový výstup do souboru `std.err`, obojí ve složce `std` u příslušného bloku v projektu, ke kterému je tento program spouštěn. Mimo složky `std` je na stejné úrovni také vytvořena složka `output`, nad kterou je vybraný program spouštěn — tedy do ní generuje výstupní soubory, či složky. Vzhledem k přesměrování pomocí funkce `chdir()` musíme také počítat s úpravou cesty ke spouštěnému souboru — tento problém je ovšem řešen již na úrovni PHP aplikace, která generuje dávkové soubory. Z pohledu tohoto souboru a adresářové struktury je třeba pouze zajistit stabilní umístění všech spouštěných programů, nevyjímaje tento program.

5.5.2 Deamon

Program *Runner* je potřeba spustit vždy, když existují dávkové soubory, které má za úkol vykonat. Tento program je možné spouštět ručně, ovšem tím by byla výrazně omezena automatická funkčnost celého systému. Stejně tak je nevhodné spouštět program ze strany PHP, neboť právě pro tento účel byla aplikace tvořena zcela samostatně, bez jakéhokoli propojení a ze strany PHP nelze spustit program tak, aby ji neblokoval. Řešení bylo nalezeno v programu, který je na serverové straně spuštěn stále, což může obnášet nezanedbatelnou část systémových prostředků. Tento program tedy musí být především úsporný z hlediska použitých funkcí a při čekání nesmí zbytečně vytěžovat systémové prostředky — tedy nelze jeho čekání řešit nekonečný *for* cyklem.

Program může mít pevně nastavený soubor, do kterého budeme ze strany PHP atomicky připisovat čísla projektů, které jsou připraveny ke spuštění. Pracovně nazvaný program *Deamon* vždy tento soubor přečte a smaže a pro každé zadané číslo spustí v paralelním vlákně vykonání programu *Runner* a jako parametr mu předá právě toto číslo. Tímto je splněn hlavní účel tohoto programu, ovšem nyní přichází na řadu uspání programu v případě, že je soubor s čísly projektů prázdný. Toto uspání musí být nastaveno na vhodnou dobu, kterou je v našem případě 1 vteřina — tato doba je dostatečně dlouhá, aby mělo uspání smysl (program neprovádí kontrolu bez přestání) a zároveň jedna z těch, které jsou pro uživatele téměř nepostřehnutelné. Ovšem volba tohoto času záleží nejvíce na tom, zda se očekávají v systému složitější projekty s dlouhou dobou běh, či četnější projekty s krátkou dobou pro

běh.

Samotné čekání je řešeno za pomoci knihovny `signal.h` a jí implementovanými funkcemi, které pomocí signálů efektivně hlídají uběhlý čas a tak umožňují implementaci výsledné funkce `sleep(seconds)` i na unixovém systému.

Kapitola 6

Implementace

Implementace rozebírá z hlavní části použité technologie a odůvodňuje, proč a kde byly uplatněny. Třídění seznamu těchto technologií podléhalo rozdělení podle důležitosti používání v základních celcích implementované aplikace. V neposlední řadě jsou u vybraných implementačních nástrojů uvedeny i důležité funkce či knihovny, které byly nosnou kostrou této aplikace.

6.1 C/C++

Programovací jazyk C vyvinuli pánové Ken Thompson a Dennis Ritchie při vývoji operačního systému Unix. Dnes je to jeden z nejznámějších jazyků, zřejmě nejčastější pro psaní systémového softwaru, ale velmi rozšířený i pro aplikace. Jazyk C patří do skupin nízkoúrovňových, kompilovaných, relativně minimalistických programovacích jazyků. Mnoho dalších moderních programovacích jazyků přebírá způsob zápisu (neboli syntaxi) z jazyka C, například Java, Perl a PHP.

Naproti tomu C++ je objektově orientovaný programovací jazyk, který vyvinul Bjarne Stroustrup a další v Bellových laboratořích AT&T rozšířením jazyka C. C++ podporuje několik programovacích stylů jako je procedurální programování, objektově orientované programování a generické programování, není tedy jazykem čistě objektovým [8].

V tomto jazyce byly implementovány oba programy pro spouštění instrukcí, předaných částí zpracovávanou v PHP. Důležitou knihovnou pro běh aplikací je knihovna `pthread`, která umožňuje zavedení nového vlákna aplikace. V neposlední řadě byly v tomto jazyce implementovány i testovací soubory, o kterých bude zmínka v následující kapitole.

6.2 PHP

PHP (PHP: Hypertext Preprocessor, původně Personal Home Page) je skriptovací jazyk určený především pro tvorbu dynamického webu. Nejčastější začlenění patří přímo do struktur jazyků HTML a XHTML. PHP lze také využít pro tvorbu desktopových či konzolových aplikací. PHP je zcela nezávislý na platformě, skripty se nemusejí upravovat, aby fungovaly na více operačních systémech a jeho velkou výhodou je podpora mnoha rozličných knihoven pro různé účely — např. zpracování textu, grafiky, práci se soubory, přístup k většině databázových systémů (mj. MySQL) apod.

Díky jednoduchosti použití, ponechání vývojáři částečné svobody v syntaxi a kombinaci vlastností více programovacích jazyků se PHP rychle stalo velmi oblíbeným jazykem [13].

Pro implementaci celého systému je klíčové, aby nebyl povolen běh PHP v bezpečném režimu, tzv. `save_mode`.

6.2.1 Framework Nette

Nette Framework je open source framework pro tvorbu webových aplikací v PHP 5. Zaměřuje se na eliminaci bezpečnostních rizik, podporuje AJAX, DRY, KISS, MVC a znovupoužitelnost kódu. Využívá událostmi řízené programování a z velké části je založen na použití komponent [14].

Byl vybrán vzhledem k dřívějším zkušenostem s prací v tomto frameworku a možností využívat předpřipravené rozhraní pro práci s databází a přijímanými parametry. Největším užitkem a hlavním důvodem pro použití tohoto nástroje je ulehčení tvorby formulářů, přehledného šablonovacího systému a v neposlední řadě zjednodušení práce s databází. Ta je tvořena samostatnou vrstvou `Database`, která aplikaci odklání od psaní samotných SQL dotazů a skládá je zcela ve své vlastní režii. Framework Nette byl použit ve stabilní verzi 2.0 s pouze minimálními úpravami uvnitř samotného frameworku.

6.3 HTML

HTML (HyperText Markup Language) je značkovací jazyk pro hypertext. Je jedním z jazyků pro vytváření stránek v systému World Wide Web, který umožňuje publikaci dokumentů na internetu. Jazyk vychází z dříve vyvinutého rozsáhlého univerzálního značkovacího jazyka SGML (Standard Generalized Markup Language). Další vývoj HTML byl ovlivněn vývojem webových prohlížečů, které zpětně ovlivňovaly definici jazyka.

Pro ukončení vývoje HTML byla připravena verze 4.01. Další vývoj psaní dokumentů měla dle W3C (World Wide Web Consortium) patřit jazyku XHTML (následovník HTML, využívající univerzální jazyk XML). Pro nedokonalost vývoje okolo XHTML se část tvůrců webových prohlížečů, jako například Mozilla Foundation, Opera Software či Apple, rozhodlo založit iniciativu WHATWG (Web Hypertext Application Technology Working Group), jejímž cílem bylo vytvořit novou verzi HTML, která se posléze začala označovat jako HTML 5 [1].

6.4 CSS

Kaskádové styly (Cascading Style Sheets) byly vyvinuty jako jazyk pro popis způsobu zobrazení jazyků HTML, XHTML nebo XML. Jazyk byl navržen standardizační organizací W3C, která zatím vydala již tři úrovně specifikace. Hlavním smyslem bylo umožnit návrhářům oddělit vzhled dokumentu od jeho struktury a obsahu. Původně to měl umožnit už jazyk HTML, ale v důsledku nedostatečných standardů a konkurenčního boje výrobců prohlížečů se vyvinul jinak [3].

HTML spolu s CSS tvoří kostru webového portálu na klientské straně. JavaScriptový klient generuje kód v těchto dvou jazycích pro zobrazení ve webovém prohlížeči. Taktéž jsou tyto jazyky potřeba pro vytvoření úvodní stránky pro přístup k celému systému.

6.5 MySQL

Aplikace vyžadovala použití databázového systému, který bude volně šiřitelný, multiplatformní, lehce ovladatelný ze strany PHP a zároveň rychlý ve zpracování požadovaných dotazů. Nabízela se možnost využití buďto MySQL nebo PostgreSQL. Nakonec bylo vybráno MySQL díky předchozím zkušenostem s jeho obsluhou.

MySQL je databázový systém, který nyní vlastní společnost Sun Microsystems a který je dostupný jak pod bezplatnou licencí GPL (GNU General Public License), tak pod komerční placenou licencí. Komunikace s databází probíhá pomocí jazyka SQL. Podobně jako u ostatních SQL databází se jedná o upravenou verzi tohoto jazyka s různými rozšířeními. Tento produkt lze instalovat jak na Linux, tak i na MS Windows, ale i na další systémy. MySQL bylo od počátku optimalizováno především na rychlost, a to i za cenu některých zjednodušení [9].

6.6 JavaScript

JavaScript je interpretovaný programovací jazyk, tvořící základ vývoje webových aplikací. Pomáhá dodat stránkám interaktivitu nebo v něm vytvořit celou aplikaci. JavaScript je na standardech založený jazyk s formální specifikací a ačkoli každý z dnešních webových prohlížečů interpretuje tuto specifikaci trochu jinak (z tohoto důvodu je těžší dosáhnout stejného výsledku pro všechny uživatele), většina webových prohlížečů používá základní funkce stejně. Při vývoji aplikací v tomto jazyce je potřeba klást důraz spíše než na vlastní použití Javascriptu na použití založeném na standardech [12].

V rámci tohoto jazyka byl využit hlavně framework jQuery, který je přímo zabudován do některých prvků frameworku Nette.

Kapitola 7

Testování

Tato kapitola má za úkol čtenáře obeznámit s testováním, které probíhalo souběžně s tvorbou aplikace. Testování webové aplikace probíhalo vždy po naimplementování potřebné funkčnosti, kdy bylo definováno vše, co musí tato část bezpečně splnit a byla definována množina věcí, které lze zadat chybně a náplní testování bylo tento seznam úloh splnit. Testování po částečných implementačních cyklech přineslo výhodu v možnosti chybné části v příštím cyklu implementace opravit. Webová aplikace byla testována jako celek po dokončení každé nezávislé části, aby se po jejím uzavření mohlo přejít na část další.

Komunikace mezi webovou aplikací a spouštěcím programem byla pevně definována a tak pro testování jednotlivých částí stačilo nahradit konečné soubory testovacími, které plnili stejný účel jako soubory, které budou generovány za běhu — bylo ovšem třeba hlídat i to, zda jsou tyto soubory také správně generovány i se všemi potřebnými vlastnostmi.

Spouštěcí aplikace byly implementovány přímo tak, aby vždy splnili část úkolu, která byla naplánována k implementaci a každá implementovaná část byla detailně prozkoušena na větším množství různých dat. Vzhledem k tomu, že tento program spouští cizí programy, bylo potřeba implementovat také sadu testovacích souborů, které vytvářejí různé způsoby chování a tak umožní vidět výsledek za předem stanovených podmínek, které vedou k určení správného chování systému.

7.1 Webová aplikace

Testování vizuálních komponent probíhalo přímo s jejich tvorbou, kdy bylo potřeba hlídat, zda ve všech prohlížečích je stejná reprezentace použitých komponent i jejich stylování. Plně navrženou funkcionalitu společně se správným zobrazením podporuje prohlížeč Google Chrome verze 18 a výš. Pokud pomineme možnost předávání souborů do formuláře přetažením myší, pak je aplikace stavěna celá tak, aby se zobrazovala ve všech používaných prohlížečích i systémem shodně. Webový portál byl v rámci testů zobrazován na operačním systému Windows 7 a Ubuntu 11 v prohlížečích Chrome verze 18, Mozilla Firefox 11 a Opera 11. Na systému Windows také ještě v prohlížeči Internet Explorer 9.

Webový portál byl testován také z hlediska serverové strany, tedy PHP. Důležitými prvky pro funkčnost na straně PHP byly funkce `chmod()`, `copy()`, `unlink()` apod., které mají možnost měnit data na serverové stanici. Tyto prvky byly implementovány s ohledem na to, aby aplikace mohla běžet jak na Unixovém serveru, tak i na stanici s operačním systémem Windows, načež obě možnosti byly otestovány. Webový portál tedy ze strany zobrazování webové prezentace není nijak omezen na použitý systém a případné funkce,

které se liší svým během podle systému byly ošetřeny, aby tento fakt braly v potaz a své chování přizpůsobily — jedná se především o formát cest k souboru, který se může lišit použitými oddělovači složek.

Pro podporu funkčnosti na straně PHP serveru je potřeba povolení možnosti nahrávat přes webový formulář i binární soubory. Tato možnost může být na některých systémech standardně zakázána a je tedy potřeba před spuštěním na serveru tento fakt zkontrolovat. Při standardním nastavení systému Ubuntu 11 bylo například potřeba smazat soubor `/etc/php5/conf.d/suhosin.ini`, který znemožňuje nahrát na server soubory přeložené na tomto systému a je instalován ve výchozí podobě PHP 5.2.

V neposlední řadě bylo taktéž testováno hledání smyček, skupin a řazení bloků ve skupině v rámci generování schématu projektu. Byla vytvořena množina testovacích souborů, které byly předkládány navrženým algoritmům a na základě výsledků byly algoritmy případně opraveny. Bylo otestováno i to, že každý algoritmus v případě špatně zadaného schématu skončí v konečném počtu kroků a nevznikne zacyklení díky zpracovávání nevhodných dat.

7.2 Spouštěč

Vzhledem ke specifickým knihovnám a funkcím jimi obsaženými je běh kód spouštěcích programů zcela jiný pod systémem Windows a Linux. Spouštění programů může být bez problémů omezeno pouze na Linuxovou distribuci, neboť programy, které bude obsluhovat, jsou dostupné primárně ve verzích pro tento systém, případně jsou pro tuto distribuci přeložitelné. Kompletní překlad a testování zdrojového kódu spouštěče tedy probíhalo na systému Ubuntu 11. Zde byl překlad aplikace prováděn za pomoci parametrů `-Wall -ansi -pedantic`, které zajišťují, že se přeloží pouze kód podléhající normě *ANSI* a to bez jakýchkoli chybových hlášení či varování. Překlad obou spouštěcích programů pak probíhal pomocí programu `g++`.

Pro překlad je také nutná knihovna pthread, jenž musí být nainstalována a zadána jako parametr při spuštění, což přiložený soubor *Makefile* vše zahrnuje.

7.3 Konfigurace pro spuštění

Z testování, jež bylo uvedeno výše, tedy vyplývá potřebná konfigurace systému pro běh webového portálu. Víme, že ze strany klienta není kladeno žádné omezení pro spuštění portálu, ovšem na straně klienta jich několik je a jim se musí serverová stanice přizpůsobit. První podmínkou je samotný operační systém, který musí být Linuxovou distribucí, kde je nainstalována trojice programů: Apache 2, PHP 5.2 a MySQL 5.5 — souhrnné označení pro tuto čtveřici je pak *LAMP* server. Pro překlad programů na serverové stanici je vyžadována implementace *POSIX Threads*.

Samotná PHP aplikace je psána pod frameworkem Nette, z čehož vyplývá, že pro běh aplikace je třeba se řídit stejným návodem uvedeným na [5], jako pro instalaci sandboxu. Dále pak jen stačí nastavit v souboru `app/config.neon` nastavit přístup k databázi a do databáze nahrát dump inicializační databáze ze souboru `sql/install.sql`.

Kapitola 8

Závěr

Závěrečná kapitola této práce obsahuje jak shrnutí dosažených výsledků, tak i možnosti dalšího vývoje webového portálu směrem k rozšíření základní funkčnosti. V první části je prezentováno vše, co se v rámci práce podařilo implementovat a jsou zde uvedeny části, které si vyžádali nejvíce času pro implementaci. Možnosti rozšíření byly uvažovány v přímé návaznosti na to, co již bylo v práci uděláno společně s tím, co návrh již podporuje a není pro toto rozšíření třeba měnit vnitřní logiku aplikace.

8.1 Dosažené výsledky

Výsledný portál pokrývá požadavky vytvořené specifikace, které rozšiřují požadavky uvedené zadáním projektu. Ve fázi návrhu byl systém doplněn o implementovaná rozšíření, čímž se s těmito úpravami počítalo již při první fázi implementace a uvažovaná rozšíření nejsou nadstavbou systému, ale přímo jeho součástí.

Při implementaci vlastního systému se vyskytly problémy, které si vyžádaly mnohem více času, než bylo v návrhu očekáváno. Z tohoto důvodu byl návrh rozdělen do více částí, kterým byla přidělena různá priorita zpracování. Největší prioritu dostala funkčnost programu pro spouštění, na který webová aplikace navazuje a poskytuje mu vizuální obslužné prostředí. Druhou prioritou pak bylo implementování základních funkcí webového portálu, mezi které patří systém správy uživatelů a systém správy bloků a projektů. Správa uživatelů byla nastavena jako priorita nerozdělitelná, neboť na ní je závislý přístup do aplikace. Správa bloků a projektů již byla rozdělena podle důležitosti akcí. Akce s větší prioritou jsou přidávání a editace a nižší prioritou pak mazání a klonování kroků, bez kterých je možné aplikaci řádně otestovat a ověřit správnou funkčnost celého systému. Nejmenší prioritu pak dostaly procesy, zabývající se nadstandardním zlepšením uživatelského komfortu, jakým je například možnost vložení souboru do formuláře přetažením.

Z důvodu pečlivého zpracovávání částí s největší prioritou nezbylo v závěrečné fázi tolik času na stejně pečlivé zpracování méně prioritních částí a tyto části již nejsou tak detailně testovány a mohou se v nich případně vyskytnout chyby. Taktéž není implementována možnost klonování a mazání projektů, ovšem obě tyto funkce jsou již navrženy tak, aby k jejich dokončení postačilo použití již implementovaných funkcí, případně jejich minimální úprava.

Ve výsledku bylo dosaženo webového portálu, který nabízí uživatelům s vyšším oprávněním nahrávat do systému vlastní spustitelné programy a ty sdílet se všemi uživateli systému. Toto nahrávání programů je zabaleno do tří základních kroků, které nabízejí intuitivní ovlá-

dání společně s možností rychlého definování všech patřičných detailů pro evidenci tohoto programu v systému. Pro všechny uživatele je pak zavedena možnost tvorby projektu, který je předkládán ve formě přehledného schématu a nabízí tak uživateli možnost mít nad svým projektem stálý přehled i při změně propojení jednotlivých kroků projektu. Viditelná část webového portálu je tedy dobře strukturovaná a vystavěna na jednoduchosti použití. Skrytá část, která obstarává spouštění projektů, byla vypracována tak, že zohledňuje možnosti paralelního zpracování a lépe tak využívá systémových prostředků.

Pro možnost úprav je výhodné použití frameworku Nette, který zavedl objektově–orientovaný přístup programování a rozdělil aplikaci podle návrhového vzoru MVC (Model View Controller). Tento přístup nabízí lepší orientaci ve zdrojovém kódu, což je důležité pro jakákoli další rozšiřování systému. Všechny vlastní funkce byly psány s ohledem na jejich možnost znovupoužití a efektivitu jejich vykonávání, což vedlo ke snížené době zpracovávání ze strany PHP. Taktéž kód spouštěcí aplikace v C++ byl psán stylem, kdy se snažíme co nejvíce samostatných celků oddělit do vlastních funkcí, což ve výsledku vedlo k dobré udržovatelnosti a opravitelnosti kódu již v dílčích fázích implementace, které probíhaly jako oprava po testování aplikace.

Vývoj podléhal především účelu, aby výsledná aplikace byla i nadále udržovatelná a nabízela možnosti další práce se všemi zdrojovými kódy a další rozšíření mohlo tyto kódy využívat. Vývoji samotné aplikace byly věnovány souhrnně dva měsíce intenzivní práce, přičemž analýza a návrh systému probíhali průběžně od vzniku zadání této práce, které se do zmíněných dvou měsíců práce nepočítají.

8.2 Možnosti dalšího vývoje

V kapitole specifikace a návrhu aplikace jsme definovali rozsah aplikace, který byl oporou pro implementaci v rámci diplomové práce, ovšem vzniklo i mnoho dalších nápadů, které by mohli systém vylepšit. Tato práce je již nad rámec projektu a může posloužit tedy i jako vize pro směr dalšího vývoje. Uveďme tedy nejdůležitější body, které je možné do systému přidat.

Při vykonávání projektu systém uvažuje s tím, že vkládané programy jsou již dostatečně otestovány a jejich spuštění probíhá bez nečekaných problémů a dále je uvažováno i to, že program běží pouze omezenou kratší dobu a uživatel nepotřebuje tuto dobu již nijak ovlivňovat. Ovšem v případě, že v systému budeme chtít poskytnout uživateli větší kontrolu nad systémem a také možnost zadávání spustitelných programů s možnou chybou za běhu, pak je vhodné dát uživateli možnost ukončit vykonávání projektu, či jediného souboru po uplynutí zadané doby. Tato funkce musí být implementována ze strany spouštěče na základě předané informace. V navrženém systému je prostor pro předání dalších informací pro spouštěcí program a nic nebrání v jednoduchém rozšiřování funkčnosti na podobné bázi. Stačí tedy pouze ke spouštěnému programu přidat položku pro zadání maximálního času pro běh a na straně spouštěče tuto informaci oddělit a použít ji, což vzhledem ke způsobu implementace spouštěče není složité — obdobně jsou již předávány jiné informace ovlivňující zpracování programu.

Webový portál taktéž umožňuje zasílání ztracených hesel či registraci pomocí zadaného e–mailu. Tuto komunikaci je však možné použít i pro potřeby zasílání zpráv o vyhotovení projektu. Do portálu je možné jako rozšíření implementovat taktéž i interní zprávy, které se budou starat o oznámení, že byl projekt úspěšně dokončen. Generování těchto správ by pak musel mít na starost samostatný skript, vykonávaný v zadaných časových intervalech a výsledky jeho činnosti by se pouze zobrazovali uvnitř systému po přihlášení, či podle

uživatelova nastavení rovnou zasílali na vyplněný e–mail. Toto rozšíření patří do rozšíření uživatelského komfortu při ovládání portálu a nebylo zahrnuto do návrhu aplikace.

Další možnosti rozšíření patří do změny konceptu celého systému, ovšem pro všeobecný přehled o možnostech implementovaného systému je vhodné je v této práci zmínit. Při dalším postupu v rozvoji aplikace je možné systém detailněji zaměřit na možnosti práce s programy, které již na serverové stanici již jsou nainstalovány a tedy využít webový portál pro využívání celé již přednastavené pracovní stanice. V tomto případě by však musel být kladen větší důraz na bezpečnost systému, neboť by nemohla být zavedena pravidla omezující přístup do jediné sekce stanice. Webový portál je nyní navržen pro univerzální použití a navržené rozhraní tak splňuje potřeby i pro takto rozšířenou verzi systému.

Literatura

[1] Castro, E.: *HTML, XHTML a CSS : názorný průvodce tvorbou WWW stránek.* Computer Press, 2007, ISBN 978-802-5115-312, 438 s.

[2] Catsoulis, J.: Fault Tolerance and Triple Modular Redundancy (TMR). [Online; navštíveno 03. 05. 2010].
URL http://www.embedded.com.au/pages/TMR.html

[3] Croft, J.: *Mistrovství v CSS : pokročilé techniky pro webové designéry a vývojáře.* Computer Press, 2007, ISBN 978-802-5117-057, 409 s.

[4] Elnozahy, E. M.; Melhem, R.; Mossé, D.: Energy-Efficient Duplex and TMR Real-Time Systems. In *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium*, Washington, DC, USA: IEEE Computer Society, 2002, ISBN 0-7695-1851-6, str. 256.

[5] Foundation, N.: Rychlý a pohodlný vývoj webových aplikací v PHP — Nette Framework. 2012, [Online; navštíveno 01. 05. 2012].
URL http://nette.org/

[6] Graphviz: Graphviz — Graph Visualization Software. 2012, [Online; navštíveno 01. 05. 2012].
URL http://www.graphviz.org/

[7] Hlavička, J.; Racek, S.; Golan, P.; aj.: Číslicové systémy odolné proti poruchám. 1992.

[8] Kadlec, V.: *Učíme se programovat v jazyce C.* Computer Press, 2005, ISBN 80-7226-715-9, 277 s.

[9] Kofler, M.: *Mistrovství v MySQL 5.* Computer Press, 2007, ISBN 978-802-5115-022, 805 s.

[10] Straka, M.: Generátor hlídacích obvodů pro komunikační protokoly Xilinx FPGA. In *Počítačové architektury a diagnostika 2007*, University of West Bohemia in Pilsen, 2007, ISBN 978-80-7043-605-9, s. 129–136.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8441

[11] Straka, M.: Aplikace hlídacích obvodů v architekturách odolných proti poruchám. In *Počítačové architektury a diagnostika 2008*, Liberec University of Technology, 2008, ISBN 978-80-7372-378-1, s. 97–102.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8698

[12] Suehring, S.: *JavaScript: krok za krokem.* Computer Press, 2008, ISBN 978-80-251-2241-9, 335 s.

[13] Vrána, J.: *1001 tipů a triků pro PHP*. Computer Press, 2010, ISBN 978-802-5129-401, 456 s.

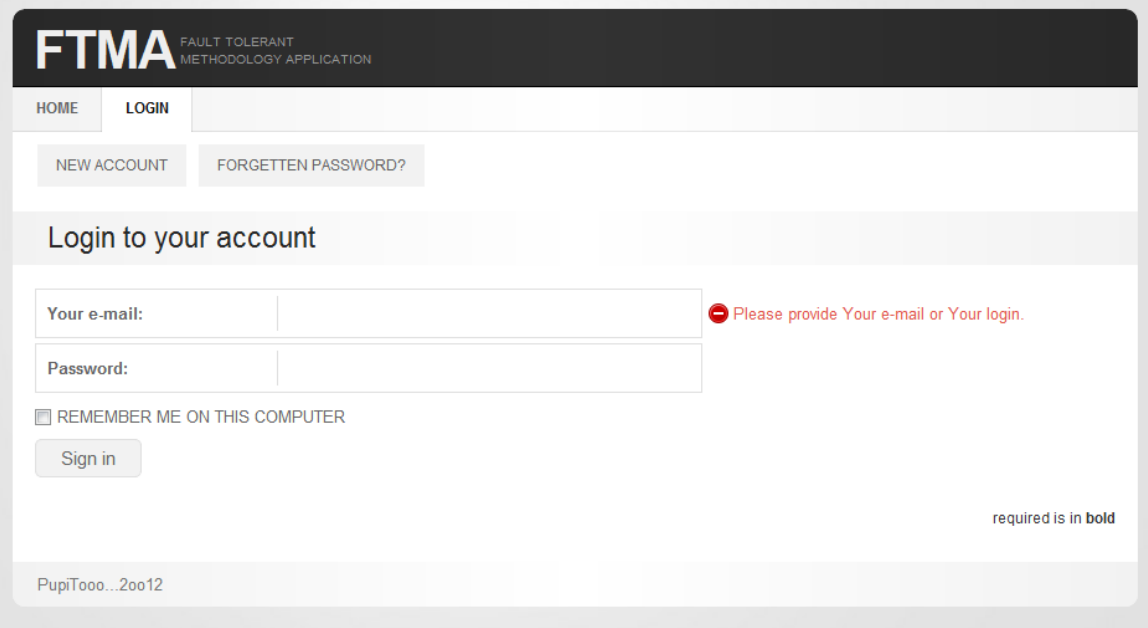
[14] Wikipedie: Nette Framework — Wikipedie: Otevřená encyklopedie. 2011, [Online; navštíveno 27. 11. 2011].
URL
http://cs.wikipedia.org/w/index.php?title=Nette_Framework&oldid=7613227

[15] Wikipedie: Scalable Vector Graphics — Wikipedie: Otevřená encyklopedie. 2012, [Online; navštíveno 12. 05. 2012].
URL http://cs.wikipedia.org/w/index.php?title=Scalable_Vector_Graphics&oldid=8078228

Příloha A

Uživatelův průvodce portálem

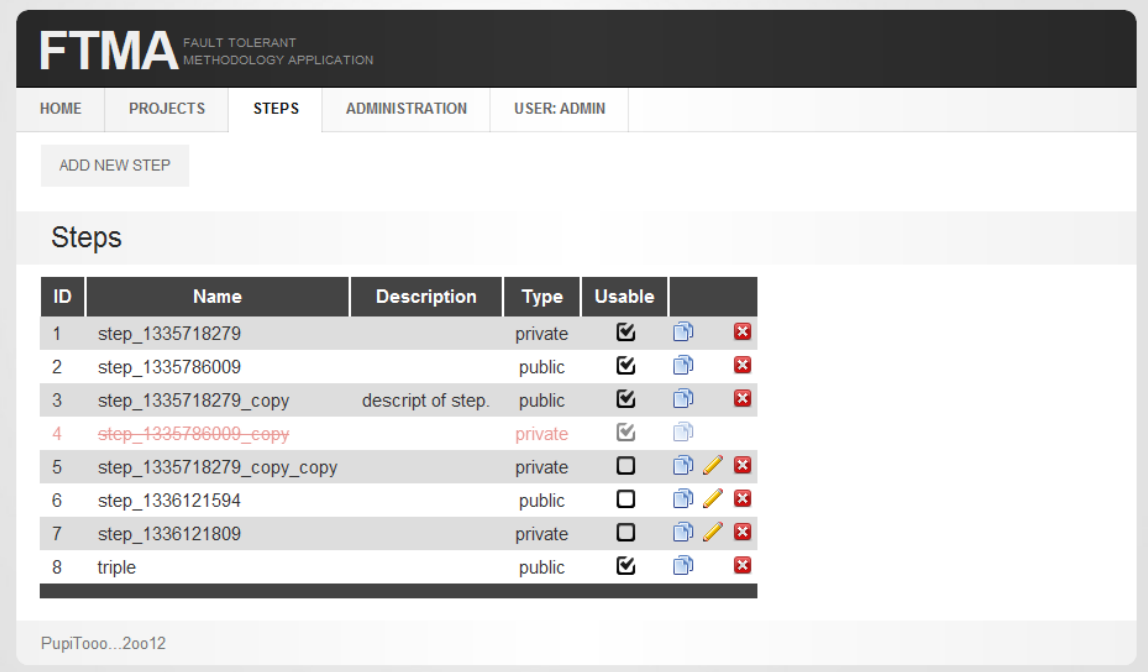
Obrázek A.1 ilustruje vzhled celé aplikace, společně s přihlašovacím formulářem pro vstup do systému. Jak je vidět vpravo od položky pro vyplnění přihlašovacího e–mailu, formuláře jsou validovány během vyplňování. Taktéž je zavedena možnost registrace do systému a zaslání zapomenutého hesla.



Obrázek A.1: Vzhled aplikace s přihlašovací obrazovkou.

Na dalším obrázku A.2 je vidět menu s položkami pro roli administrátora. Ostatní role nemají přístup do některých sekcí a toto menu obsahuje méně položek. Na tomto obrázku je taktéž vypsán seznam všech bloků, vytvořených přihlášeným uživatelem. Tlačítka všech akcí pro zvolený řádek seznamu jsou označeny obrázkem ilustrujícím tuto akci a opatřeny komentářem, popisujícím tuto akci.

Nejdůležitější akcí je samotné přidávání bloku. To je rozděleno do tří kroků, což je naznačeno za pomoci tří otisků nohy v pravé horní části obrazovky na obrázku A.3 (na obrázku je znázorněn první ze tří kroků, což napoví i komentář u těchto otisků). Na tomto obrázku je dále vidět, že lze přidat program do formuláře pouhým přetažením souboru do vyznačené oblasti. Po přetažení se tyto soubory vypíší pod touto oblastí společně s jejich



Obrázek A.2: Výpis vytvořených bloků.

velikostí.

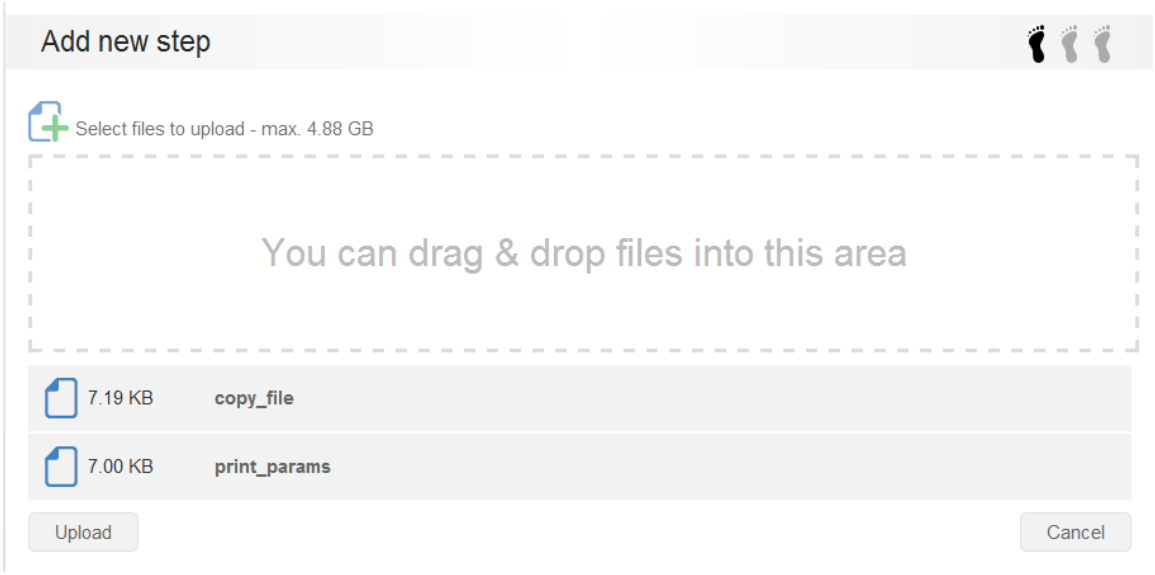
Druhý krok přidávání bloku je znázorněn na obrázcích A.4 a A.5. Na prvním z nich vidíme, že jsou programy vizuálně odděleny a každý z nich může být okamžitě z bloku odebrán po kliknutí na černý kříž v pravém horním rohu příslušného bloku. Každý blok je opatřen v levé střední části úchytem (šipky nahoru a dolu), jehož přetažením docílíme přetažení celého bloku, a tím i změny pořadí, v jakém se programy budou v rámci bloku spouštět. V případě nahrávání souboru jako hodnoty příslušného parametru je zobrazen i fakt, zda byl již soubor nahrán, jak je vidět u prvního parametru druhého bloku (nápis „UPLOADED“ v černém poli).

Obrázkem A.6 je znázorněn výpis projektů. Ty jsou v systému členěny na vlastní projekty, které lze spouštět, a na veřejné, ze kterých lze jen klonovat projekt mezi vlastní projekty.

Při vytvoření nového projektu nejdříve musí dojít k vložení kroků do projektu. To lze provést uchopením a přetažením zvoleného bloku do oblasti, ve které se nacházejí všechny již zahrnuté bloky. Na obrázku A.7 jsou bloky zařazené do projektu v pravé části a lze je také z projektu vyřadit pomocí tlačítka pro smazání. Bloky, ze kterých lze vybírat jsou tříděny podle vlastnictví vlevo. Taktéž je na tomto obrázku vidět, že název projektu lze upravit po kliknutí na titulek stránky — změna je uložena po kliknutí mimo tento titulek.

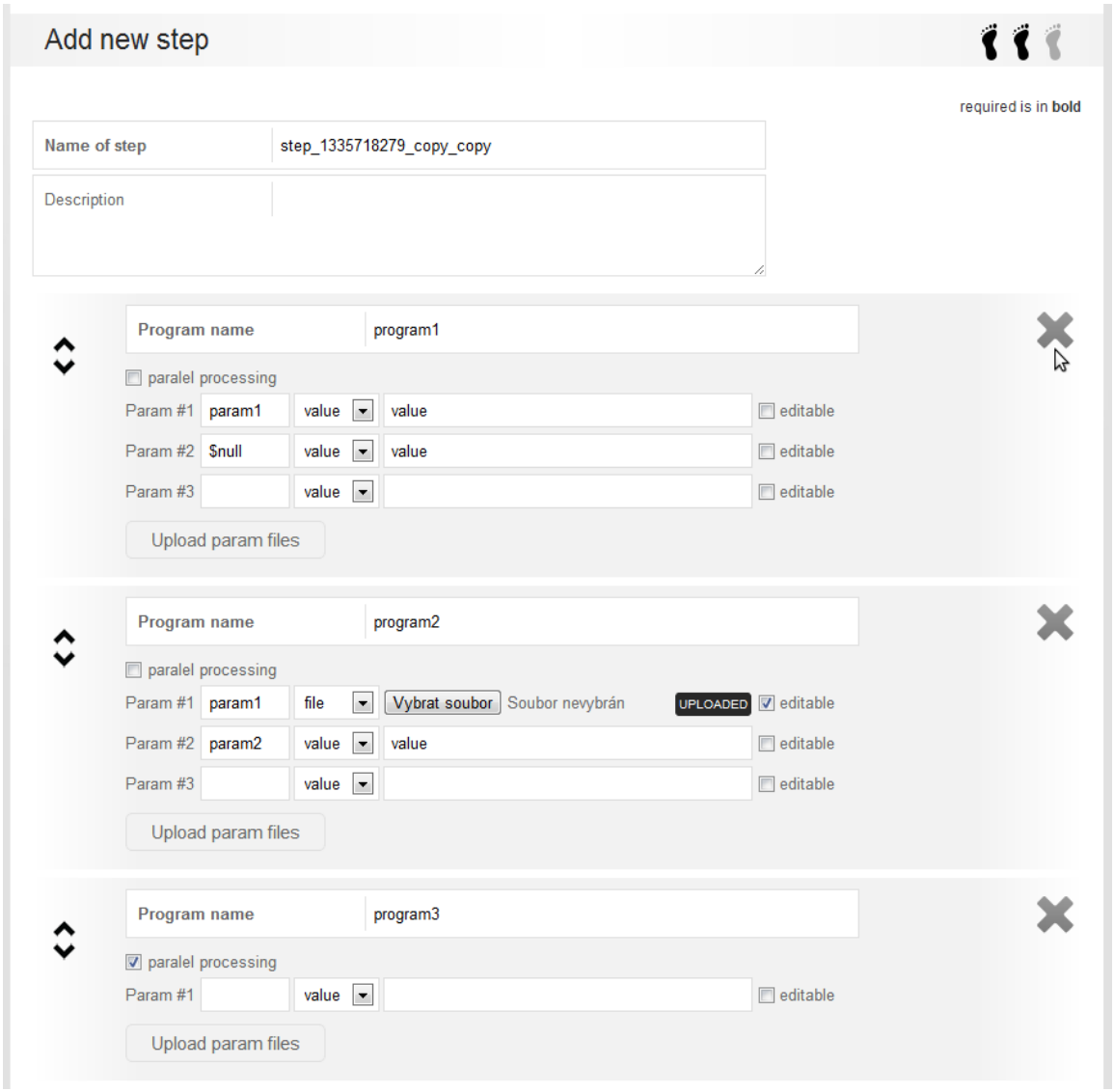
Hlavní obrazovkou tvorby projektu je samotné zobrazení schématu — ukázáno obrázkem A.8. Zde je vidět rozvržení všech bloků patřících do projektu. Po kliknutí na název bloku (ten je vždy v prvním řádku bloku a je ohraničen dvojitými šipkami) lze přistoupit k editaci vybraného bloku. Po dokončení úprav můžeme projekt spustit.

Po úspěšném spuštění projektu je zobrazena stránka uvedená obrázkem A.9, na které je již finální výpis spuštěného projektu. Tento výpis je členěn podle bloků, které jsou v projektu zahrnuty. U každého bloku je vypsán standardní výstup a chybový výstup, společně s nabídnutou možností stáhnout archiv s vygenerovanými soubory (samozřejmě v přísluš-

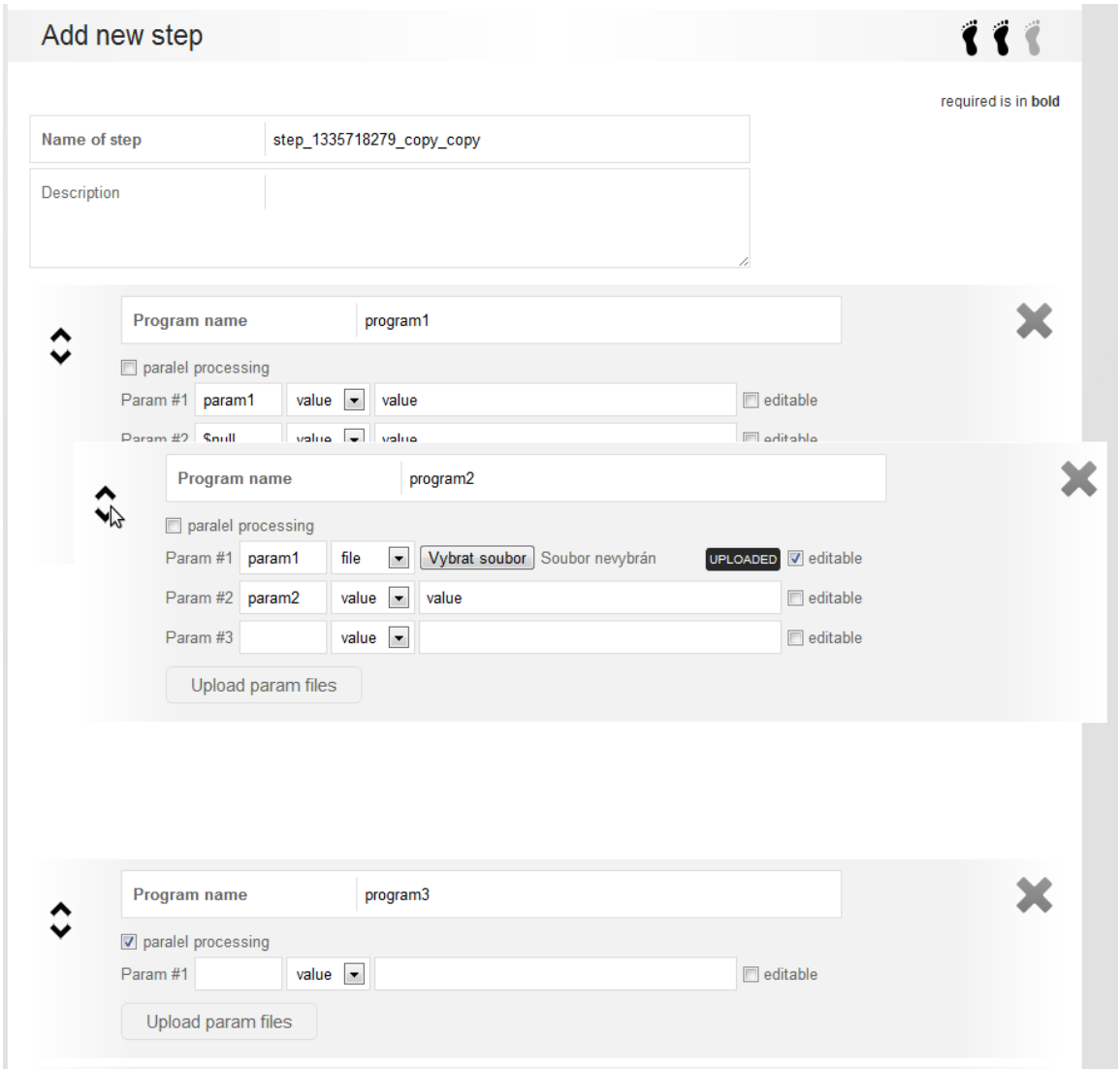


Obrázek A.3: Přidávání programů do bloku.

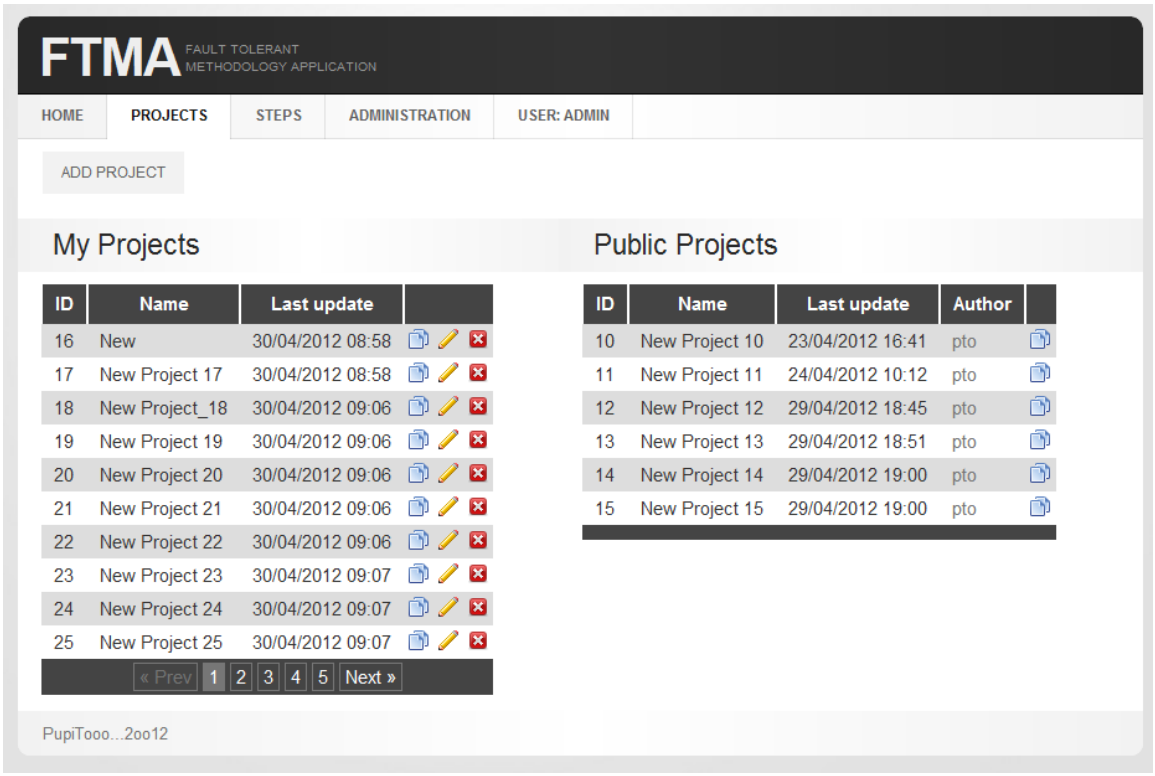
ných složkách), pokud nějaké byly vygenerovány.



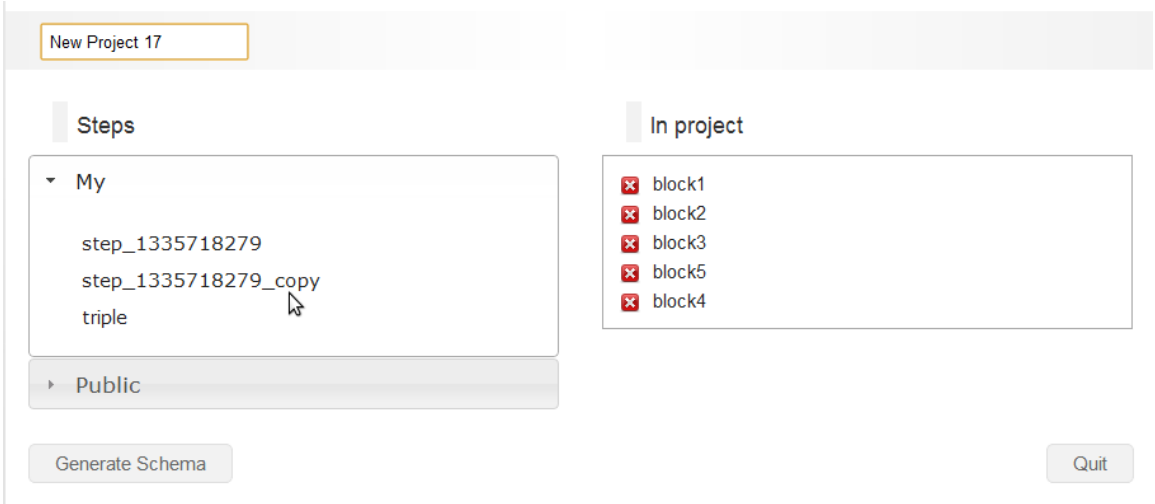
Obrázek A.4: Editace parametrů v bloku.



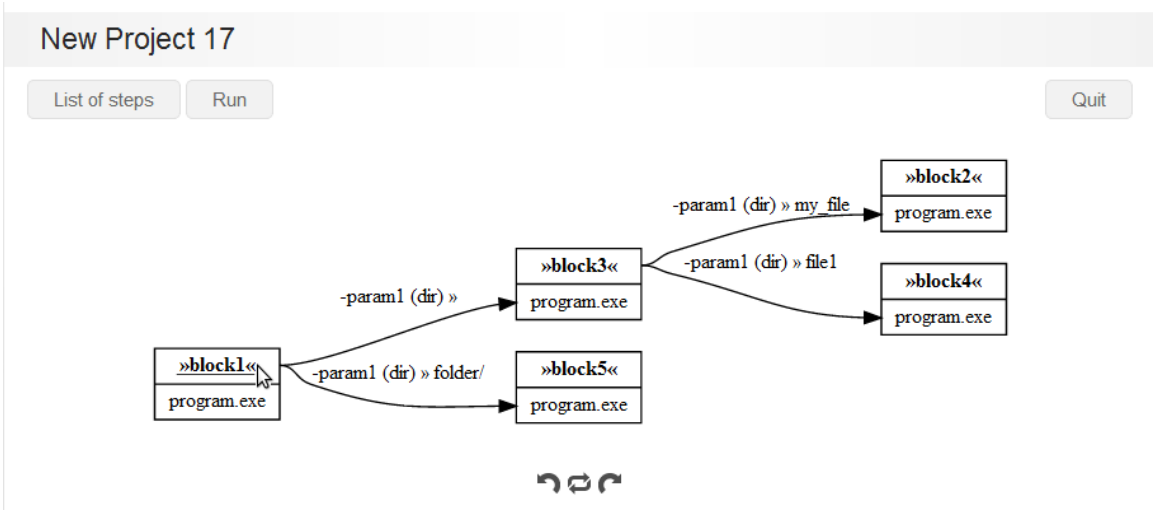
Obrázek A.5: Posun programů v bloku.



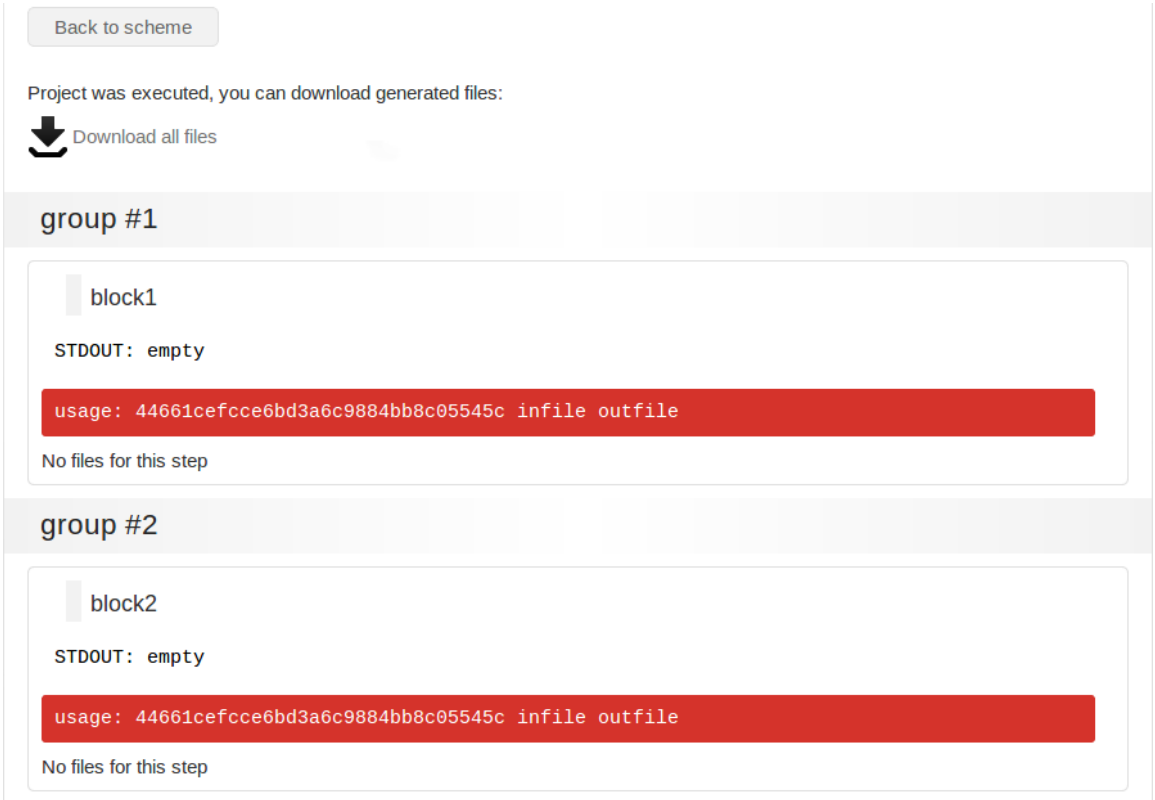
Obrázek A.6: Výpis veřejných a vlastních projektů.



Obrázek A.7: Vzhled aplikace s přihlašovací obrazovkou.



Obrázek A.8: Schéma projektu.



Obrázek A.9: Výstup po spuštění projektu.

Příloha B

Obsah CD

Obsah CD je následující:

- Zdrojové kódy aplikace.
- Testovací programy.
- Technická zpráva ve formátu PDF.
- Technická zpráva ve formátu LaTeX.
- Soubor README.